НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Рудик М.Р.
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Розробка програмного забезпечення для ведення обліку використання енергоресурсів на об'єктах виробництва на платформі ОС Windows»

Виконав:
студент IV курсу, групи ТР-61
Рудик Микита Романович _____

Керівник:
професор, д.т.н.
Отрох Сергій Іванович _____

Рецензент:
старший викладач, к.т.н
Зенів Ірина Онуфріївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
„Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020 р.

ЗАВДАННЯ
на дипломну роботу студенту

Рудик Микита Романович
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка програмного забезпечення для ведення обліку використання енергоресурсів на об’єктах виробництва на платформі ОС Windows

керівник роботи професор, д.т.н. Отрох Сергій Іванович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1268-с**

2. Строк подання студентом роботи 10 червня 2020

3. Вихідні дані до роботи: Мова програмування Java, система управління базами даних MySQL, платформа створення візуального інтерфейсу JavaFX, комп’ютер із операційною системою Microsoft Windows.

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс,

зробити висновки за результатами роботи

5. Перелік ілюстраційного матеріалу

2. Аналіз предметної області та доступних систем обліку енергоресурсів. 3. Засоби розробки програмного продукту. 4. Опис програмної реалізації. 5. Модель взаємодії користувача із програмною системою.

6. Дата видачі завдання ” 1 ” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	01.10.19	
2.	Вивчення та аналіз задачі	12.02.20	
3.	Розробка архітектури та загальної структури системи	23.04.20	
4.	Розробка структур окремих підсистем	30.04.20	
5.	Програмна реалізація системи	04.05.20	
6.	Оформлення пояснювальної записки	11.05.20	
7.	Захист програмного продукту	15.05.20	
8.	Передзахист	10.06.20	
9.	Захист	17.06.20	

Студент _____ Рудик М.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Отрох С.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Обсяг дипломної роботи складає 58 сторінок, 29 рисунків. 3 додатки і 10 посилань на використану літературу

Метою дипломної роботи є розробка програмного забезпечення для ведення обліку використання енергоресурсів на об’єктах виробництва на платформі Windows

В процесі розробки було використано об’єктно-орієнтовану мову програмування Java. Інтерфейс програми написаний на мові XML за допомогою візуального засобу Scene Builder. Розробка програмного продукту велась в середовищі Intelij IDEA із застосуванням збірника проектів Apache Maven.

Результатом роботи стало створення програмної системи, що забезпечує зберігання інформації про палива та їх витрату, а також оснащена функціоналом створення звітів по витратах, як загальних, так і деталізованих.

Ключові слова: облік, алгоритм, паливо, витрата, розробка, система управління базами даних.

ABSTRACT

The amount of the thesis is 58 pages, 29 pictures. 3 additions and 10 references to the used literature

The purpose of project is develop software to manage of using energy resources on the Windows platform

The java programming language was used in the development process. The program interface is written in XML with using the visual tool Scene Builder. The software product was developed in the Intelij IDEA environment using the Apache Maven project collection.

The result of software system creation that provides storage of information about fuels and its consumption, as well as equipped with the functionality of creating cost reports, both general and detailed.

Keywords: accounting, algorithm, fuel, consumption, development, database management system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Кроссплатформеність – здатність програмного забезпечення виконувати свої функції в декількох різних платформах (операційних системах);

АСКОЕ – автоматизована система комерційного обліку енергоресурсів;

JVM – Java Virtual Machine;

Java SE – Java Standart Edition;

JDK – Java Development Kit;

SQL – Structured Query Language;

JDBC – Java DataBase Connectivity;

Драйвер – програма, що керує зовнішніми елементами системи;

ВСТУП

Енергетичні ресурси – джерела різних видів енергії, доступні для використання об’єктами виробництва та побутовими користувачами.

Підвищений попит на різноманітні енергоресурси, а також ріст закупівельних цін на більшість видів палива звернули увагу, в першу чергу, промислових підприємств на недосконаліть використовуваних на сьогодні систем обліку та контролю за витратами енергоресурсів. Проблемою більшості промислових виробництв в поточних умовах стало стрімке застарівання, сформованих ще в умовах планової економіки, принципів та технічних рішень обліку енергоресурсів, які до сьогодення дійшли майже без змін. Використання подібних рішень все частіше призводить до надмірних втрат енергії, а отже, в умовах ринкової економіки, і до фінансових збитків підприємства.

Зі сторони підприємств або об’єктів виробництва, енергоресурси, наприклад, паливо, розглядається як товар, тобто об’єкт системи купівлі-продажу, до якого, як і до всіх інших товарів, мають застосовуватися правила комерційного обліку та, за можливістю, оптимізації витрат.

Метою дипломного проекту є створення автоматизованої системи обліку використання енергоресурсів на об’єктах виробництва, а саме створення додатку для операційної системи Windows, функції якого дозволяли б користувачу відслідковувати тенденцію до використання того, чи іншого палива із можливістю створення звітів по витратам.

Програмна реалізація описаної вище системи представлена додатком на платформі JavaFX, що дозволяє створювати інтерфейсно навантажені додатки та програмні продукти. Основною мовою програмування для системи використана мова Java версії 1.8. Для побудови архітектури база даних була обрана система управління базами даних MySQL із робочим простором MySQL Workbench та віртуальним сервером MySQL

Server. Для з’єднання сервера бази даних та створюваної системи використано коннектор сімейства MySQL Connectors JDBC Driver for MySQL.

Підключення бібліотеки драйверу до системи, а також автоматизація збірки проекту у файл формату “executable jar” виконане за допомогою фреймворку Apache Maven, що забезпечує декларативну збірку програмного продукту на основі файлу специфікацій “pom.xml” чим значно спрощує підключення та використання у програмному коді сторонніх бібліотек та інших ресурсів, а також позбавляє від необхідності збирати проект системи вручну, чим зменшує вірогідність помилок під час збірки.

Безпосередньо, написання програмного коду створеного додатку виконувалося у інтегрованому середовищі розробки InteliJ Idea на основі парадигми об’єктно-орієнтованого програмування.

1. ЗАДАЧА РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВЕДЕННЯ ОБЛІКУ ВИКОРИСТАННЯ ЕНЕРГОРЕСУРСІВ НА ОБ'ЄКТАХ ВИРОБНИЦТВА НА ПЛАТФОРМІ ОС WINDOWS.

Метою розробки є створення програмного забезпечення, що надає функціонал та інструментарій для реєстрації витрат та ведення обліку використання енергоресурсів на об'єктах виробництва, як приватних підриємствах, так і об'єктах державного господарювання.

Призначенням даного програмного продукту э надання користувачу повного функціоналу для адміністрування бази даних, яка зберігає дані про витрати енергоматеріалів на цільовому об'єкті, представлення описуваного функціоналу в простому табличному вигляді користувацького інтерфейсу, коректна оборобка і представлення інформації із бази даних та аналіз витрат енергоресурсів та грошових коштів на об'єкті господарювання.

Програмний засіб було розроблено для платформи Microsoft Windows XP/7/10, проте дана програмна реалізація системи може бути запущена на будь-якій операційній системі із наперед встановленим пакетом JRE, що дає можливість запускати виконувані JAR-файли не напряму через операційну систему, а через JVM.

Вхідними даними для системи є інформація із бази даних, що описує використовувані палива, а також, безпосередньо, витрати описуваних палив. Таблиці заповнюються користувачем через візуальний інтерфейс.

Вихідні дані системи представлені автоматично створеними параметризованими звітами по використанню палив на цільовому об'єкті господарювання. Створюються як загальний звіт в якому представлена лише сумарна фінансова витрата, так і деталізований звіт, який надає інформацію по витратах кожного із використовуваних у вказаному місяці палив.

Необхідні можливості, якими має бути забезпечений створюваний програмний модуль:

- Підлкючення до бази даних;
- Формалізація введених користувачем даних;
- Обробка даних про використання енергоматеріалів;
- Створення та зберігання звітів про енерговитрати за місяць;
- Підрахунок відсотку, що складає використання певного палива від загальної маси витрат;
- Аналіз динаміки зміни витрат енергоматеріалів;
- Можливість створювати як загальний, так і деталізований звіти.

Можливі додаткові функції:

- Редагування записів в таблицях без видалення самих записів;
- Наявність простого інтерфейсу користувача;
- Фільтрація даних по певному параметру;
- Виставлення параметру пріоритетності записів задля спрощеної навігації по записам таблиць;

Користувачами створеної системи можуть бути як державні виробництва, так і юридичні особи, що являють собою об'єкти виробництва, які вважають за необхідне провести модернізацію принципів обліку та контролю за використанням енергоресурсів.

2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСТУПНИХ СИСТЕМ ОБЛІКУ ЕНЕРГОРЕСУРСІВ

2.1 Опис предметної області

Ведення обліку та керування використання енергоресурсів – це процес послідовного спостереження, відслідковування та аналізу даних про енерговитрати підприємства чи об’єкту господарювання. Для здійснення подібної діяльності необхідні ресурси для збору, накопичення, формалізації, зберігання та аналізу інформації. На сьогодні більшість подібних функцій передається під керування автоматичних систем комерційного обліку енергоресурсів.

Автоматизована система комерційного обліку енергоресурсів – комплекс програмно-технічних рішень, що забезпечує постійний моніторинг, облік та контроль за використанням енергоресурсів. Призначення автоматизованої системи комерційного обліку енергоресурсів визначається як, підвищення рівня енергоефективності об’єкту господаювання за рахунок оптимізації енерговитрат. Крім того автоматизовані системи обліку енергоресурсів, налаштовані на контроль за використанням електричної, та теплової енергії можуть бути складовими комплексів «розумний будинок».

Основними цілями впровадження АСКОЕ є:

- Автоматичний аналіз енергоспоживання об’єкту господарювання;
- Скорочення та спрощення процесу збору та аналізу інформації задля підвищення оперативності прийняття рішень по зміні енергетичної політики підприємства;
- Спрощення процесу формування рахунків для проведення розрахунків із постачальниками енергоресурсів;
- Визначення місць крадіжок енергоресурсів;

- Підвищення енергоефективності організації за рахунок зниження втрат енергоматеріалів;
- Виключення людського фактору при обліку використання енергоматеріалів;

2.2 Існуючі рішення

Автоматизована система комерційного обліку енергоресурсів “Пульсар” (рисунок 2.1.1) являє собою комплекс апаратно-програмних засобів, що взаємодіють між собою, дозволяючи проводити енергоаудит об’єктів господарювання, та будувати політику енергофективності для конкретного виробництва чи домашнього господарства.

Апаратна частина комплексу являє собою комплекс датчиків, лічильників та виконавчих механізмів та зовнішніх систем керування. Підключення вимірювальних приладів та систем керування виконується за рахунок контролерів.

Вимірювальними пристроями можуть бути більшість лічильників та витратомірів, що представлені на ринку. Так само і зовнішні пристрої- регулятори температури, реле тиску чи газовими конекторами.

Програмне забезпечення комплексу складається із власної розподіленої бази даних та засобів її роботи та взаємодії із програмним шаром логіки.

Сама база даних може бути встановлена як на фізичний сервер, так і розгорнута у вигляді хмарного сервісу із доступом через веб-інтерфейси.

Також в комплексі “АСКОЕ Пульсар” реалізована система керування доступом.

Тобто доступна можливість налаштування адміністративного та користувацького доступу до даних та можливостей керування системою.

Рисунок 1.2.1 Схема комплексу “АСКОЕ Пульсар”

Вихідні дані можуть надаватися у вигляді графіків, діаграм та звітів, що дає користувачу можливість аналізувати стан енергосистеми організації та вчасно вживати заходи задля покращення енергоефективності підприємства.

Також, в комплексі програмної системи передбачений монітор надзвичайних ситуацій, який своєчасно повідомить користувача про виникнення потенційно небезпечної ситуації.

Основними недоліками вищеописаного комплексу енергоуправління є досить непростий інтерфейс користувача, а також проблеми синхронізації деяких даних, що може призводити до збоїв у роботі системи. Також існує проблема із неможливістю синхронізувати дані із лічильників та датчиків старих моделей, тобто система чітко працює тільки із сучасними цифровими варіантами лічильників, датчиків, тощо.

Крім того, для здійснення керування енергоспоживанням об’єкту господарювання в автономному режимі, автоматизована система комерційного обліку енергоресурсів “Пульсар” виковує достатньо складний комплекс апаратних засобів керування, від датчиків і лічильників, до автоматизованих реле тиску та перемикачів, що, в свою чергу, веде до додаткових витрат при впровадженні даної системи, що пов’язані із необхідністю встановлення, налаштування, технічного обслуговування та поточного ремонту апаратної частини. Зважаючи на складність та об’ємність апаратних засобів даної системи, обслуговування всієї апаратної частини є доволі складним та об’ємним процесом, який вимагає наявності технічних спеціалістів високої кваліфікації.

2.3 Висновки до розділу

В даному розділі було розглянуто функціонал автоматизованих систем комерційного обліку енергоресурсів, їх особливості та цілі впровадження. Також була проаналізована система комерційного обліку енергоресурсів “АСКОЕ Пульсар”, що представляє собою комплекс апаратно-програмних рішень, створених задля

автоматизації контролю, обліку, та оптимізації використання енергоносіїв, а, відповідно і оптимізації витрат на їх закупівлю.

Перевагами використання даної системи на промислових підприємствах є:

- Автоматичність обліку використання енергоресурсів;
- Система контролю та попередження про нештатні ситуації
- Можливість автоматизації використання енергоресурсів за рахунок системи апаратних засобів (лічильників, реле, програматорів, контролерів, перемикачів, тощо);
- Функціонал для автоматичного конструювання звітів;
- Гнучкість системи та широкі можливості по налаштуванню алгоритмів контролю, моніторингу та керування;

Незважаючи на достатньо вагомі переваги, система “АСКОЕ Пульсар” має і не менш вагомі недоліки, серед яких:

- Висока вартість як самої системи в цілому, так і її компонентів;
- Велика кількість апаратних та програмних вузлів, що збільшує кількість; “точок відмови”, що в свою чергу, веде до більшої вірогідності несправності чи некоректної роботи;
- Необхідність в наявності кваліфікованих спеціалістів для обслуговування апаратного комплексу системи;
- Громіздкість системи;
- Складний візуальний інтерфейс користувача, що вимагає навчання персоналу для роботи із системою;

3. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

3.1 Мова програмування Java

Java (версії 1.8) – мультиплатформенна об’єктно-орієнтована мова програмування, що була розроблена компанією Sun Microsystems (в наступному і донині, компанією Oracle). При компіляції, додатки, створені даною мовою перетворюються транслятором в спеціальний байт-код, який може запускатися і працювати на будь-якій платформі і комп’ютерній архітектурі, для якої існує реалізація віртуальної машини JVM

Основними властивостями та перевагами мови програмування Java є:

- Можливість створення програмного забезпечення на одній платформі і запуск його на будь-якій іншій платформі із попередньо встановленою JVM;
- Можливість створення програм, що працюють у веб-режимі в браузері та безпосередньо веб-додатків;
- Можливість створення багатофункціональних і ефективних додатків для мобільних платформ, мікроконтролерів, серверів, бездротових модулів, датчиків, а також практично для будь-яких інших електронних пристроїв;
- Широкий спектр інструментарію для створення програмного забезпечення на основі багатьох розповсюджених архітектур, а також налагодження взаємодії та комунікації між усіма модулями створюваного програмного продукту;
- Можливість створення систем та алгоритмів будь-якого ступеню складності та об’єму, завдяки чому мова Java набула широкої сфери використання як в невеликих нішевих проектах, так і в великих корпоративних програмних продуктах або як основна мова програмування, або як надійний та зручний засіб для створення окремих модулів системи;

Вибір мови Java як основного інструменту створення дипломного програмного продукту обгрунтований широкими можливостями для реалізації взаємодії створюваної системи із базою даних, можливістю організації простого та зрозумілого для користувача інтерфейсу, а також детальною документацією, що надає компанія Oracle через свої ресурси в режимі вільного доступу.

3.2 JDK

JDK (англ. Java Development Kit) – середовище розробки Java – комплект розробника програмних продуктів на мові програмування Java, що включає в себе компілятор Java, стандартні бібліотеки класів Java, технічну документацію, а також середовище виконання JRE. Структура побудови середовища розробника JDK зображено на схемі (рисунок 3.2.1).

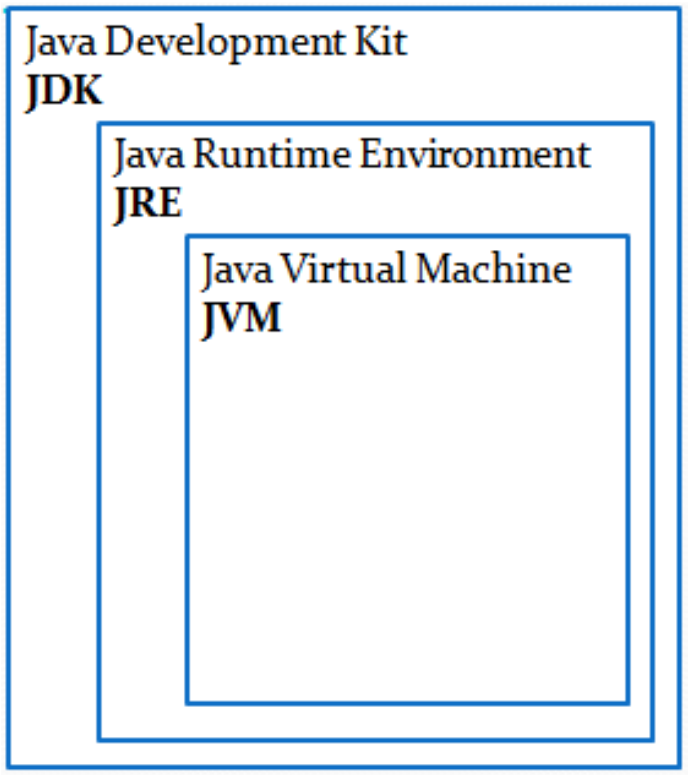


Рисунок 3.2.1 Структура JDK

3.3 JVM

JVM (англ. Java Virtual Machine) – віртуальна машина Java – ключовий компонент платформи, який безпосередньо виконує опрацювання байт-коду, який створюється із вихідного коду програми, компілятором. Програми, що призначені для виконання на віртуальній машині Java, повинні бути скомпільовані в стандартизованому двійковому форматі, який, представляється у вигляді файлів із розширенням “.class”. Так, як проект може складатися із декількох класів, то, для полегшення та уніфікації зберігання великих програм і файлів розширення “.class”, програма може бути упакована в збірку із розширенням “.jar”. Крім того, за допомогою спеціальних інструментів-пакувальників, програму можна запакувати у виконавчу збірку “executable jar”, яка, по своїй суті, є самодостатньою і може бути запущена на будь-якому комп’ютері із встановленою віртуальною машиною Java. Архітектура JVM на базі Java SE сьомої версії представлена на схемі (рисунок 3.3.1).

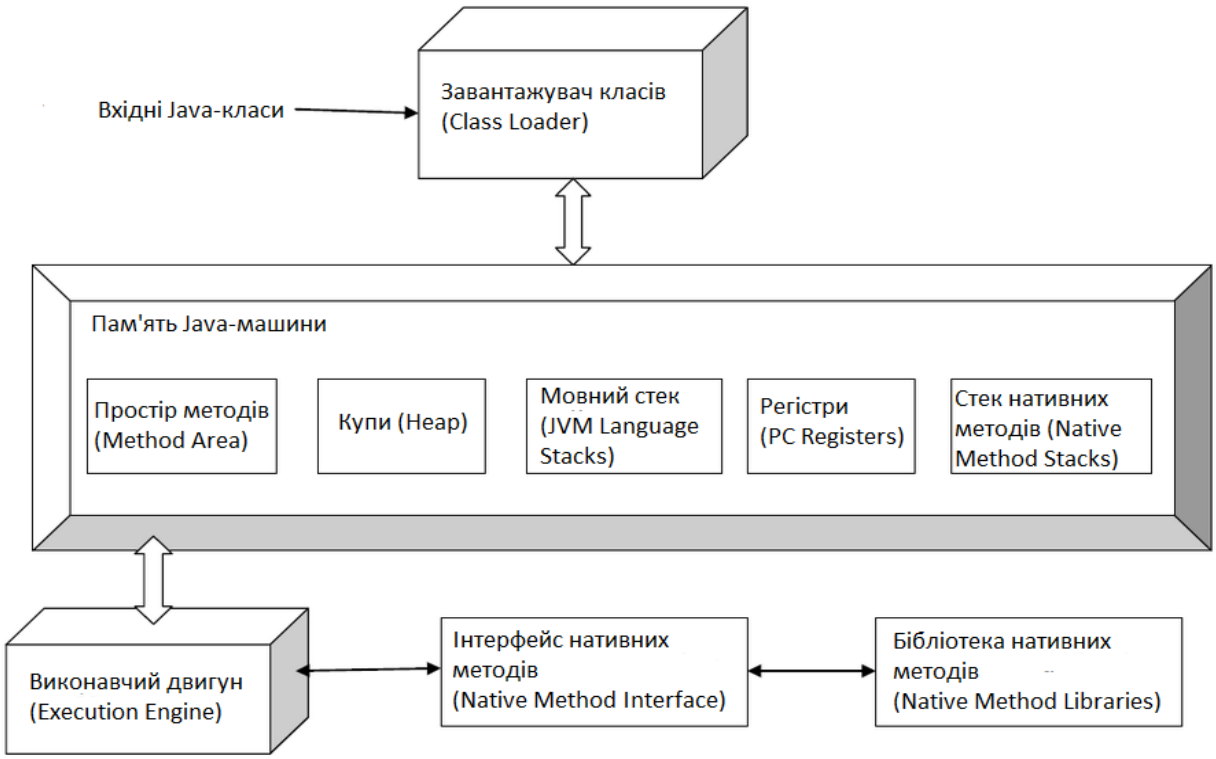


Рисунок 3.3.1 Архітектура JVM на базі Java SE 7.

3.4 Середовище JRE

JRE (англ. Java Runtime Environment) – середовище виконання Java – мінімальна реалізація віртуальної машини Java, яка необхідна для запуску та виконання Java додатків. JRE складається із віртуальної Java (JVM) і бібліотеки початкових Java-бібліотек (якщо додаток використовує сторонні Java-бібліотеки, то вони повинні поставлятися разом із збіркою програми і в самій програмі обов'язкове встановлення залежностей для кожної із сторонніх бібліотек.

Середовище JRE розповсюджується на основі ліцензії GNU GPL, яка дає можливість користуватись, модифікувати та розповсюджувати програмний продукт, в тому числі і на комерційній основі. Також дана ліцензія гарантує всім користувачам розповсюджуваного програмного продукту всі вищезазначені права.

Базування розповсюдження JRE на основі вищеописаної ліцензії дає можливість авторам, додавати пакет середовища JRE безпосередньо до збірки свого програмного продукту, що значно спрощує впровадження та експлуатацію створюваної системи.

3.5 Платформа JavaFX

JavaFX – платформа на основі мови програмування Java, що використовується для створення додатків із складним та навантаженим візуальним інтерфейсом. Можливе використання JavaFX не тільки для програмних продуктів, що запускаються безпосередньо з комп'ютера користувача, а також для мобільних і веб-додатків.

Структура класів JavaFX представляється в вигляді графу, що представлений на схемі (рис 3.5.1).

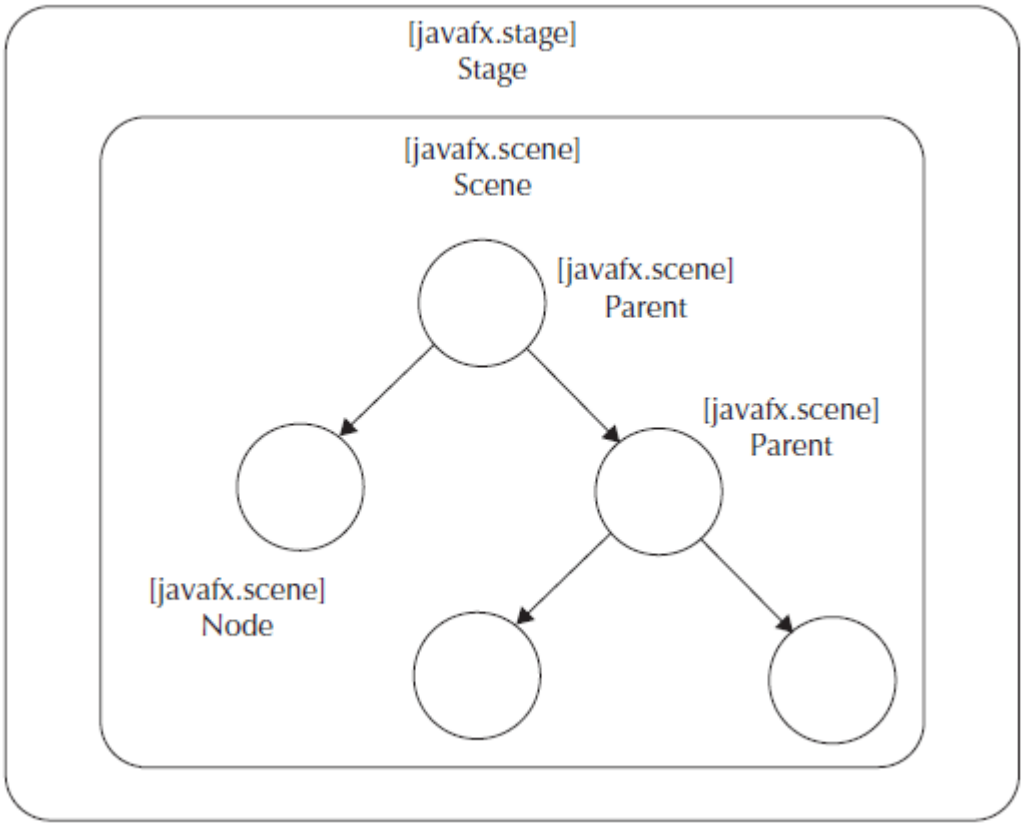


Рисунок 3.5.1 Структура класів JavaFX

Ключовими класами структури JavaFX є:

- Клас javafx.stage є контейнером усіх наступних елементів візуального інтерфейсу. Основним призначенням даного класу є визначення вигляду графічного інтерфейсу, який залежить від конкретної реалізації класу Stage для конкретної платформи, на який запускається додаток;
- Клас javafx.scene є наслідником класу javafx.stage, тому зберігає у собі всі наступні графічні елементи. Проте, на відміну від батьківського класу, зберігає графічні елемента всередині об’єкту Stage у вигляді графа Stage.Graph;
- Клас Parent виконує функції зберігання, обробки та видалення наступних об’єктів класу Node, які в свою чергу є основними вузлами інтерфейсу.

- Клас Node представляє собою, власне, клас, реалізацією якого вузол-елемент інтерфейсу, який безпосередньо взаємодіє із користувачем. Клас Node є абстрактним класом, реалізацією якого є вищеописаний клас Parent;

Загальна ієрархія класів, що реалізують абстрактний клас Node має вигляд наступного представлення (рисунок 3.5.2)

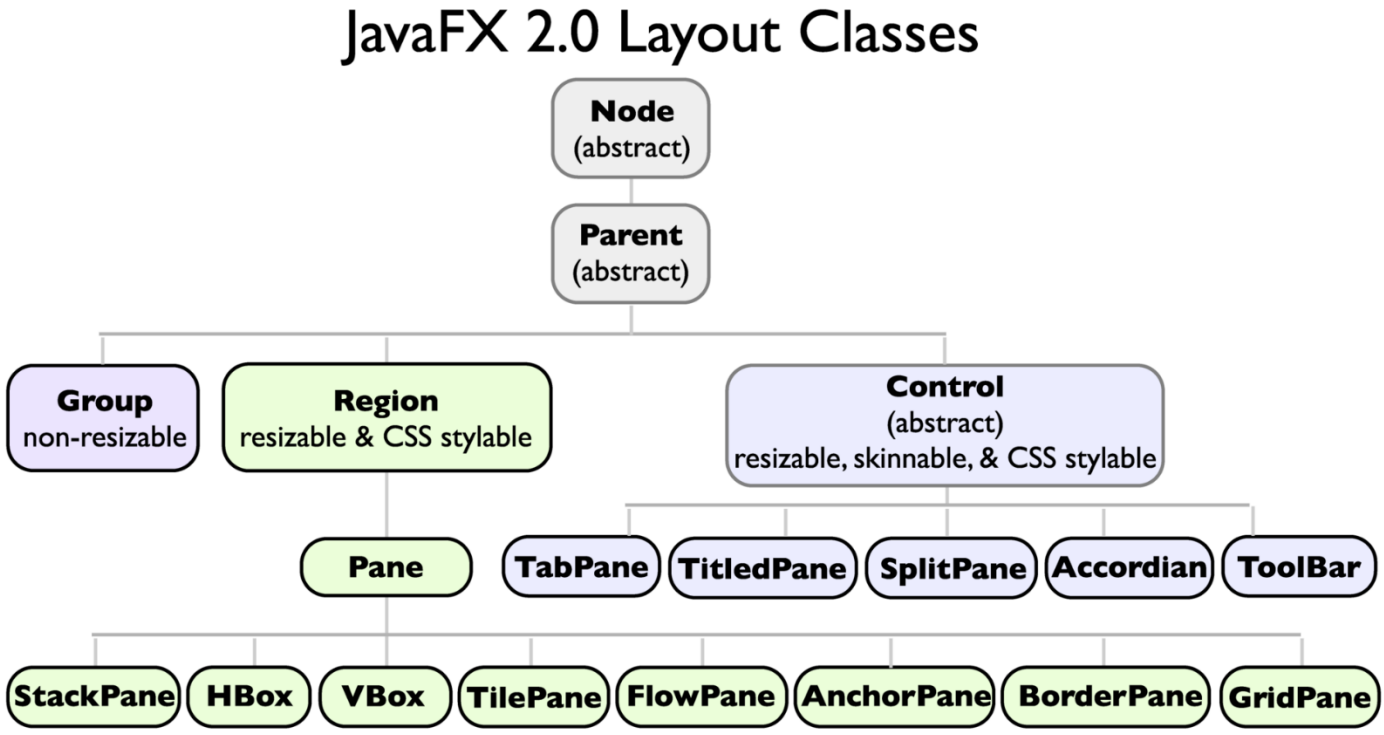


Рисунок 3.5.2 Ієрархія реалізацій класу Node.

3.6 Мова програмування баз даних SQL

SQL (англ. Structured Query Language) – мова структурованих запитів – декларативна мова програмування, що використовується для організації, створення, модифікації та управління базами даних в реляційних базах даних. SQL являється інформаційно-логічною мовою програмування, задача якого є описання, створення, додання, зміна, перегляд та вилучення даних із керованої бази даних, тобто дана мова

програмування не вважається тьюринг-повною (не має можливості реалізації будь-якої обчилювальної функції).

Серед переваг мови SQL можна виділити такі як, незалежність від конкретної системи управління базами даних, наявність чітких стандартів роботи з мовою, декларативність, тобто розробник лише декларує дані, якими необхідно маніпулювати і спосіб маніпуляції, а виконання запиту на прикладному рівні перекладається безпосередньо на систему управління базами даних.

Традиційним і головним недоліком мови SQL вважають складність, так як метою створення даної мови було створення максимально простого засобу роботи з базами даних, але в процесі розвитку технології, мова настільки ускладнилась, що перестала відповідати початково закладеним принципам простоти та інтуїтивності.

3.7 Простір баз даних MySQL

MySQL – система управління реляційними базами даних, створена компанією Sun Microsystems (в подальшому і до сьогодення, компанією Oracle) на мові C/C++ і адаптована для платформ Microsoft Windows, Linux, macOS, FreeBSD, Solaris. Архітектура MySQL дозволяє зберігати цілі числа довжиною до восьми байтів, строкові значення фіксованої та змінної довжини, числа з плаваючою та фіксованою точкою, а також підтримує всі стандартні запити та модифікатори мови SQL.

Значною перевагою MySQL перед схожими системами управління базами даних є більш висока швидкодія виконання запитів, яка досягається завдяки реалізації функціоналу MySQL мовою C/C++, яка відрізняється більшою швидкістю через свою низькорівневість і роботу напряму із пам’яттю. Проте, не зважаючи на мову реалізації, бази даних, створені в середовищі MySQL добре синхронізуються та взаємодіють із додатками, створеними мовою Java завдяки драйверу JDBC.

3.8 MySQL Workbench

MySQL Workbench – інструмент із пакету інструментів MySQL який поєднує створення, програмування, моделювання та підтримку бази даних в об’єднане робоче середовище, завдяки чому надає можливість проектування бази даних у візуальному режимі, що значно спрощує процес адміністрування бази даних.

Для розробки програмного продукту було обрано MySQL Workbench Community Edition, яка надає наступний спектр можливостей:

- Наявне представлення модель бази даних в графічному вигляді;
- Наявний та функціональний механізм встановлення зв’язків міє базами даних таблиці, в тому числі зв’язку “багато до багатьох” із автоматичним створенням додаткової таблиці зв’язків;
- Зручний редактор SQL-запитів, що дозволяє напряму відправляти їх серверу і в реальному часі отримувати відповідь у вигляді таблиці;
- Редагування даних таблиці напряму в візуальному режимі;

3.9 Драйвер JDBC

JDBC (англ. Java DataBase Connectivity) – підключення до баз даних на Java –

мультиплатформенний стандарт взаємодії Java-додатків із різними системами управління базами даних, заснований на концепції драйверів, які реалізують підключення до бази даних завдяки спеціально ініціалізованій адресі сервера бази даних, яку називають строкою підключення до бази даних.

Архітектура драйверів побудована таким чином, що вони можуть довантажуватися у програму динамічно, тобто під час роботи, автоматично реєструватися і рекурсивно викликати самого себе коли програма викликає строку підключення, за яку відповідає драйвер.

У мові Java, драйвер JDBC реалізований в бібліотеці “java.sql” та реалізує наступі інтерфейси:

1. java.sql.DriverManager – інтерфейс, що забезпечує завантаження драйверів та створення з’єднань із базою даних. Даний інтерфейс є базовим і відповідає за ініціалізацію драйверу для конкретної бази даних
2. java.sql.Connection – інтерфейс, що визначає в цілому з’єднання з базою даних, тобто його характеристики, а також стан.
3. java.sql.Statement – інтерфейс, що виконує функції зберігання SQL-виразу, під яким мається на увазі текст запиту, параметри запиту та стан виразу.
4. java.sql.ResultSet – інтерфейс, що виконує функції контейнера для набору даних, які були отримані в результаті виконання SQL-запиту.

Схема взаємодії драйверу JDBC із базою даних, розміщеною на сервері, представлена на схемі (рисунок 3.9.1).

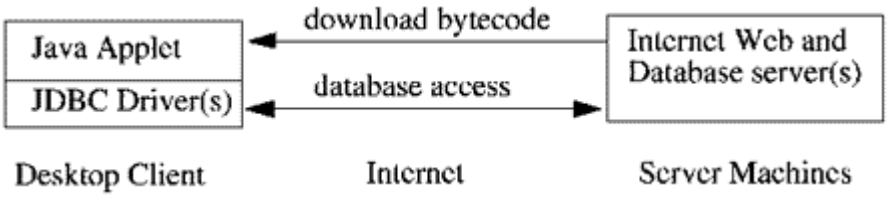


Рисунок 3.9.1 Схема взаємодії JDBC із сервером бази даних

Перевагами драйверу JDBC вважають:

- Простота розробки, тобто розробник не зобов’язаний досконало знати специфіку бази даних, якою він маніпулює.
- Відсутність додаткового клієнту для підключення до бази даних, що значно спрощує підтримку програми, та не навантажує її на рівні місця, що програма займає на комп’ютері користувача.

- Уніфікація програмного коду додатка, що позбавляє необхідності редагування взаємодії додатка і бази даних у випадку переходу на іншу базу даних
- Автоматизація підключення, тобто з’єднання з базою даних на рівні розробника обмежується ініціалізацією строки підключення.

3.10 Фреймворк Apache Maven

Apache Maven – сторонній фреймворк-збірник із відкритим вихідним кодом, призначення якого полягає в перетворенні програмного проекту в готову самодостатню збірку на основі декларативного описання проекту.

Робота Apache Maven заснована на принципі архетипів-шаблонів, кожен із яких визначений паттерном, який визначає методику побудови похідного шаблону. Однією із реалізацій принципа архетипів є автоматична побудова стандартного дерева каталогів (рисунок 3.3.2) в проекті при визначенні його як Maven-проекту.

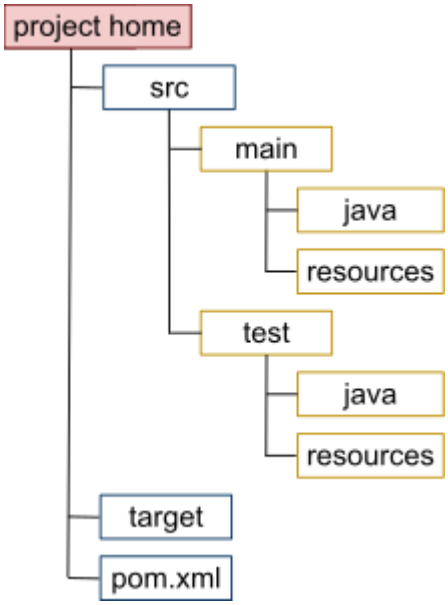


Рисунок 3.10.2 Стандартне дерево каталогів Maven-проекту

Архітектура Maven (plugin-архітектура) побудована таким чином, що дозволяє застосувати до поточного проекта необхідні плагіни, створені під певні задачі лише вказавши їх у конфігураційному файлі “pom.xml”, без потреби явно їх інсталювати, що значно зменшує об’єм пам’яті, який в подальшому знадобиться для розгортання збірки на комп’ютері користувача. Також архітектура Maven дозволяє імпортувати сторонні бібліотеки до проекту, лише імпортуючи їх в якості залежностей у файл конфігурацій, без необхідності фізичного додавання файлів в проект, більш того, Maven автоматично вирішує конфлікти версій доданих бібліотек за допомогою алгоритму вибору найновішої версії.

Основними перевагами використання Apache Maven, як менеджера та збірника проекту є:

- Простота в підключенні сторонніх плагінів та бібліотек
- Автоматична збірка проекту в виконуваний файл
- Автоматичне вирішення проблеми конфлікту версій між залежностями
- Автоматичне відключення від збірки невикористаних залежностей, тобто плагіни чи бібліотеки, які не викликаються в коді програми, не будуть включені в кінцеву збірку.

3.11 Середовище розробки Intelij IDEA

InteliJ IDEA – мультимовне інтегроване середовище розробки програмного забезпечення, що розробляється компанією JetBrains. Дане середовище надає широкий список інтегрованих інструментів для зручного процесу розробки, рефакторинга коду програми та розробки навантаженого графічного користувацького інтерфейсу.

Інструментарій та дизайн InteliJ IDEA орієнтовано на те, щоб користувач міг більше уваги приділяти розробці алгоритму створюваної програми, тому що, більшість

звичних операцій, як то контроль синтаксису, імпортування необхідних бібліотек та навіть створення шаблонних методів, виконуються середовищем автоматизовано.

Основними перевагами використання InteliJ IDEA є:

- Висока ступінь автоматизації процесу програмування
- Автоімпорт біблиотек
- Широкий набір інструментів розробки та рефакторинга

Головними недоліками даного середовища називають високі системні вимоги, та доволі великий об’єм використовуваної оперативної пам’яті, що може значно уповільнювати роботу комп’ютера користувача.

3.12 Графічний редактор Scene Builder

Scene Builder – графічне середовище-редактор для спрощення процесу проектування та побудови користувацького інтерфейсу віконного додатку.

Scene Builder надає наступний інструментарій проектування користувацького інтерфейсу:

- Додавання та видалення всіх доступних компонентів із ієрархії класів платформи JavaFX (рисунок 3.5.2);
- Редагування та налаштування доданих візуальних компонентів у режимі реального часу;
- Наявне представлення створюваного віконного інтерфейсу;
- Можливість візуального редагування вже наявних інтерфейсних вікон, представлених у вигляді файлів із розширенням “.xml”;
- Повна інтеграція у середовище розробки програмного забезпечення Intelij IDEA із можливістю запуску безпосередньо із робочого простору та вказанням редагованого файлу.

Вибір Scene Builder в якості засобу початкового проектування та редагування віконного інтерфейсу обгрунтований інтуїтивно-зрозумілим інтерфейсом даного програмного засобу (рисунок 3.12.1), можливістю вибору будь-якого із доступних для JavaFX графічних елементів та повним функціоналом для їх налаштування. Також вагомою перевагою даного редактору є можливість оперативного редагування вже створених інтерфейсних елементів із автоматичним записом проведених змін у відповідний текстовий файл.

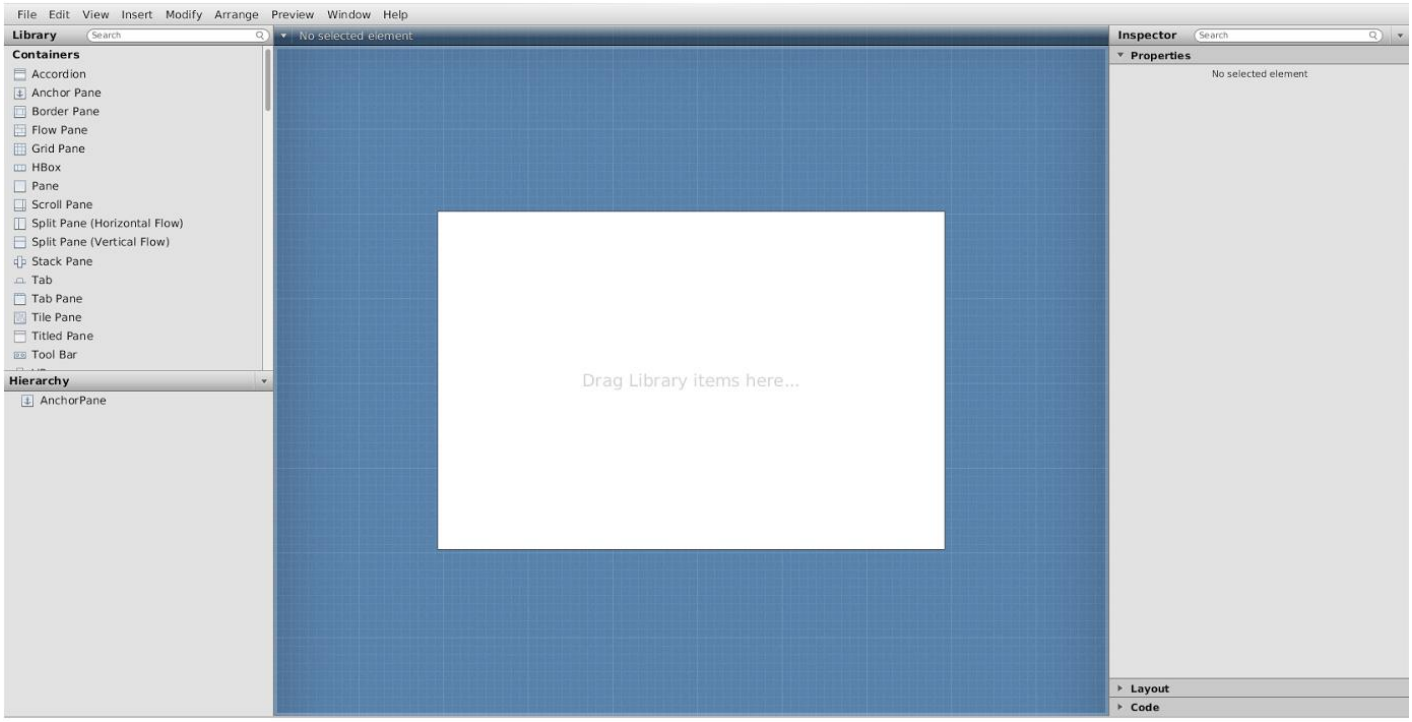


Рисунок 3.12.1 Головне вікно Scene Builder

3.13 Висновки до розділу

В даному розділі були розглянуті технології, платформи, сервіси та програми, що були використані для створення кінцевого програмного продукту.

Для програмування системи було обрано мову програмування Java версії 1.8, яка є об’єктно орієнтовною мовою програмування, а також містить достатньо широкий

інструментарій для створення програмних систем будь-якого рівня складності тобто забезпечує весь необхідний функціонал, як для конструювання користувацького інтерфейсу, так і для написання алгоритмічної частини програмної системи.

Архітектура бази даних побудована на основі системи управління базами даних MySQL та об’єднаного середовища MySQL Workbench.

Для з’єднання сервера бази даних та програмної системи використано коннектор сімейства MySQL Connectors JDBC Driver for MySQL.

Підключення бібліотеки драйверу до системи, а також автоматизація збірки проекту у файл формату “executable jar” виконане за допомогою фреймворку Apache Maven, що забезпечує декларативну збірку програмного продукту на основі файлу специфікацій “pom.xml” чим значно спрощує підключення та використання у програмному коді сторонніх бібліотек та інших ресурсів, а також позбавляє від необхідності збирати проект системи вручну, чим зменшує вірогідність помилок під час збірки.

Даний комплекс обраних програмних засобів надав всі можливості та інструментарій, що були необхідними для створення програмної системи.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Структура програмного продукту

Для загальної структури віконного додатку було обрано стандартну, автоматично сгенеровану, структуру Maven-проекту, яка в конкретному випадку має наступний вигляд (рисунок 4.1.1):

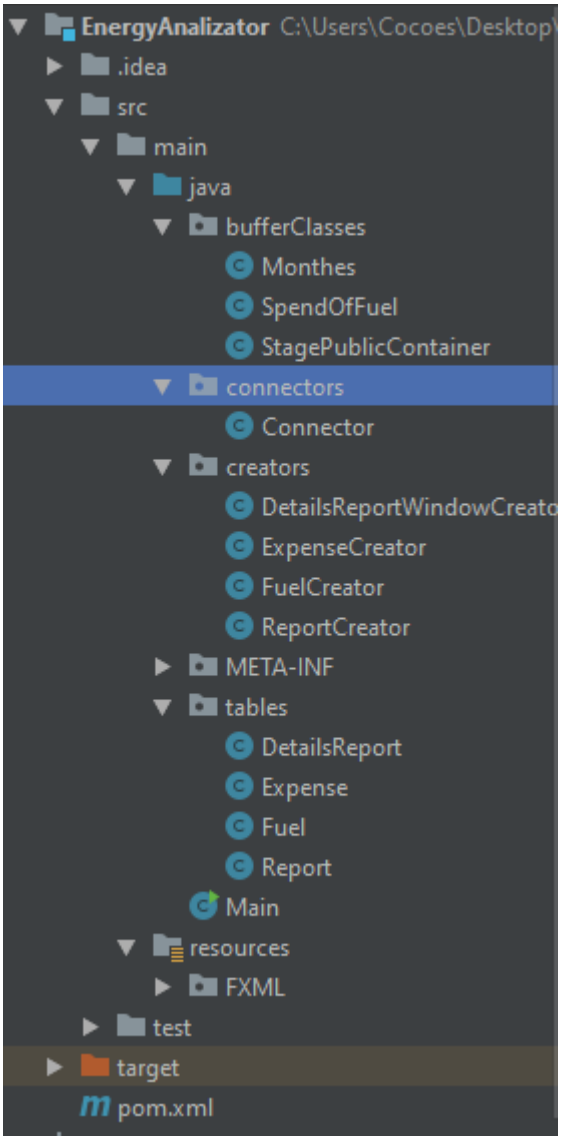


Рисунок 4.1.1 Структура программного продукту

- Каталог “EnergyAnalizator” являє собою кореневий каталог проекту, в якому розташовуються основні компоненти програми.
- Каталог “src/main/java” – директорія, де зберігаються файли вихідного коду на мові Java
- Підкаталог “src/main/java/bufferClasses” зберігає допоміжні класи, що використовуються в класах реалізації бізнес-логіки.
- Підкаталог “src/main/java/connectors” є контейнером для класів-конекторів, які забезпечують відкриття, реалізацію та закриття підключень до бази даних.
- Підкаталог “src/main/java/creators” – директорія реалізації основної бізнес-логіки програми, так як в ній зберігаються класи-обробники, які взаємодіють із візуальним інтерфейсом користувача, зчитують, оброблюють та передають інформацію до бази даних. Також дані класи оснащені функціоналом для обробки помилкових випадків та сповіщення користувача про некоректність його роботи із системою
- Підкаталог “src/main/java/tables” є контейнером для класів, що аналогічні таблицям у використовуваній базі даних та застосовуються системою як класи-посередники між візуальним представленням та базою даних.
- Каталог “src/main/recources/FXML” – каталог із іншими файлами, які використовуються під час роботи програми, в даному випадку цей каталог зберігає файли текстового представлення користувацького інтерфейсу у форматі “.xml”.
- Каталог “target” – додаткові та допоміжні файли, створювані Maven в процесі роботи
- Файл “pom.xml” – конфігураційний файл, що визначає параметри збірки, та підключення додаткових бібліотек за допомогою Maven.

4.2 Структура бази даних системи

База даних системи представлена реляційною базою даних, яка складається із трьох зв’язаних таблиць (рисунок 4.2.1).

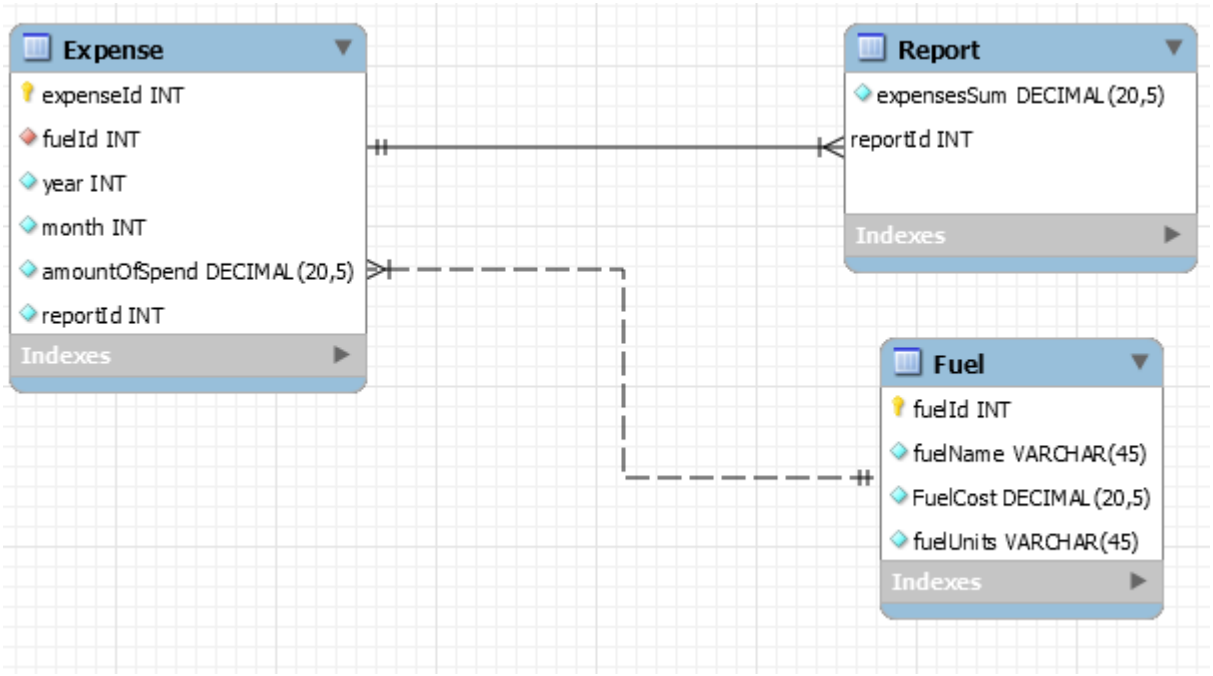


Рисунок 4.2.1 Візуальне представлення бази даних системи

Таблиця “Fuel” зберігає дані про конкретне паливо, що використовується на виробництві.

Таблиця “Report” зберігає параметризовані узагальнені звіти про матеріальні витрати на палива на підприємстві за місяць.

Таблиця “Expense” відповідає за зберігання даних витрати одного палива протягом одного місяця.

Зв’язки таблиць:

- Зв’язок: таблиця “Expense” – таблиця “Fuel” визначено як зв’язок типу “багато до одного” де батьківською є таблиця “Fuel”, тобто один запис із таблиці “Fuel” відповідає декільком записам із таблиці “Expense”;

- Зв’язок: таблиця “Report” – таблиця “Expense” визначено як зв’язок типу “один до багатьох” із батьківською таблицею “Expense”, отже один запис таблиці “Report” пов’язаний із декількома записами в таблиці “Expense” через поле reportId.

Дана таблиця пов’язана таблицею “Report” зв’язком типу “багато до одного”, тобто одному ідентифікатору звіту відповідає декілька ідентифікаторів витрат. Також існує зв’язок типу “один до одного” із таблицею “Fuel”, тобто одному запису про витрату відповідає один запис про паливо.

4.3 Структура інтерфейсу користувача

Інтерфейс користувача представлений таблицями, які є візуальним представленням таблиць бази даних та інструментами для роботи з ними, які безпосередньо взаємодіють із класом-конектором.

Головне вікно (рисунок 4.3.1) програми представлене візуальною реалізацією таблиці “Report” де в формалізованому вигляді представлені загальні дані про місячні звіти по витратах палив. Крім того дане вікно оснащене функцією фільтрування записів по вказаному у відповідному текстовому полі параметру, також даний інтерфейс надає можливість створення нового звіту, який буде автоматично сформований із записів про витрати того місяця, який буде вказаний користувачем у відповідному тектовому полі у форматі “xxxx.xx”, де ліва частина представляє номер року, а права – номер місяця.

Крім того, на головному вікні представлені кнопки – посилання, активація яких викликає відкриття інтерфейсу для роботи із відповідною таблицею у новому вікні.

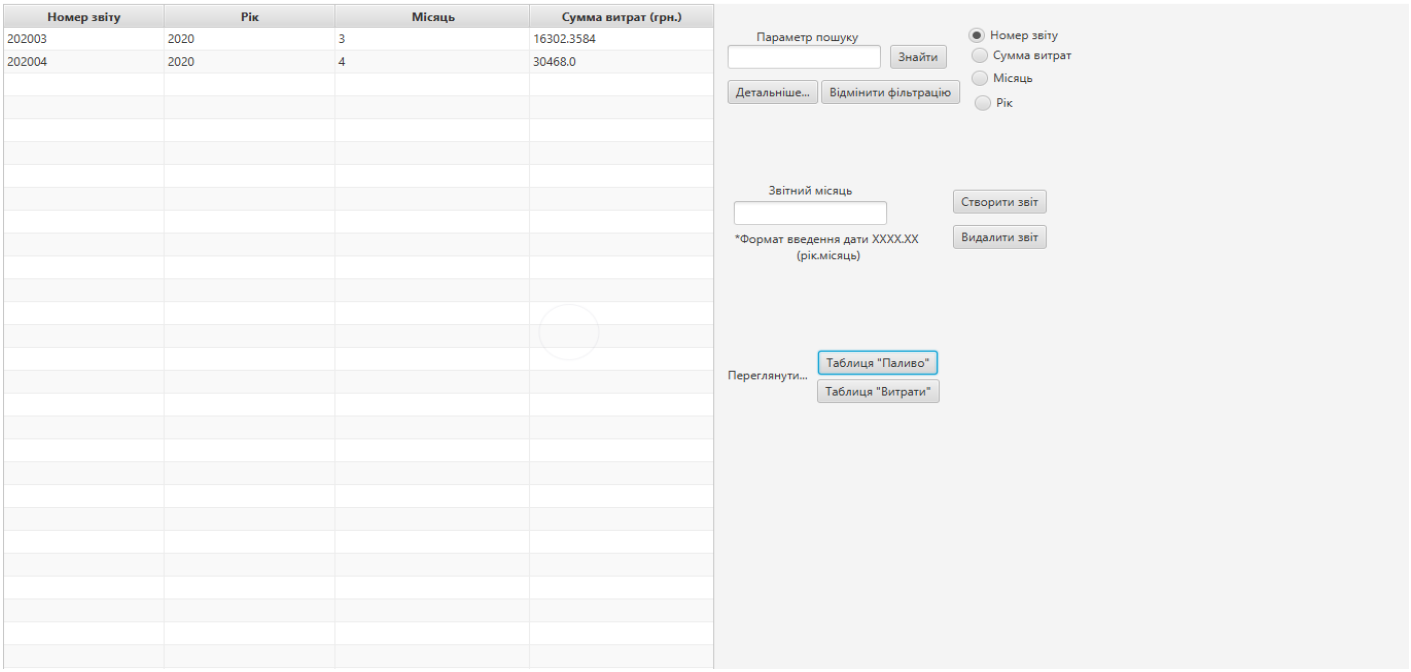
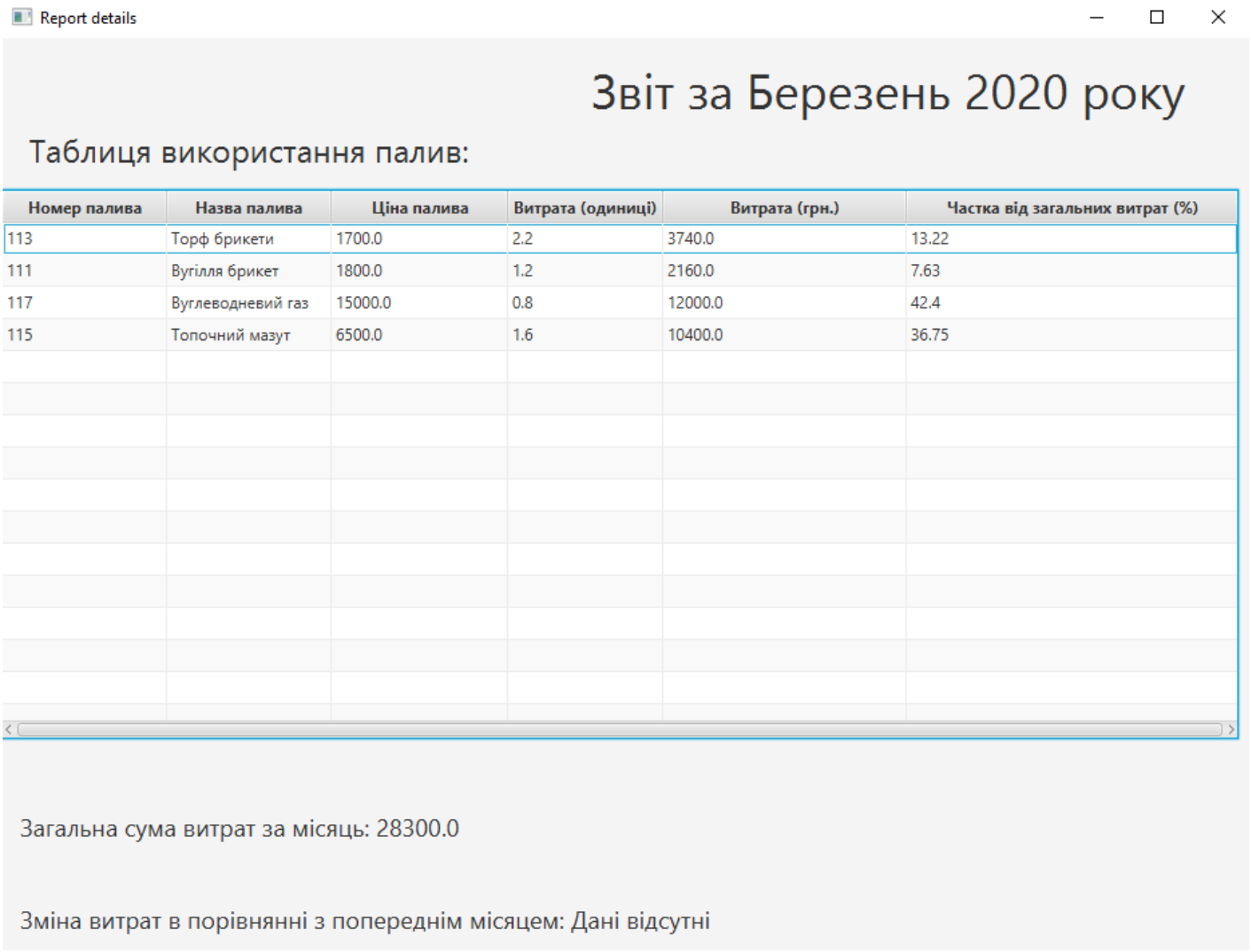


Рисунок 4.3.1 Головне вікно програми

Натискання кнопки “Детальніше” відкриває нове вікно із розширеною інформацією про обраний звіт (рисунок 4.3.2).

У вікні “Report details” деталізованого представлення обраного звіту, представлені дані про ідентифікатор, назву та ціну використаного палива, кількість витраченого палива в умовних одиницях, фінансовий еквівалент витрат палива, що вираховується шляхом сумування добутків цін використовуваних палив на кількість їх витрат, а також представлена відсоткова частка витрат кожного конкретного палива в загальній масі матеріальних витрат. При вирахуванні масової частки витрат, для уніфікації та узагальнення даних, використовуються дані саме про матеріальні витрати підприємства при використанні кожного конкретного палива у вказаних об’ємах.

Також у вікні деталізації звіту представлена сума грошових витрат на використані палива за місяць і доступна функція відслідковування зміни цих витрат через порівняння витрат поточного місяця із даними минулого і вирахування зміни у відсотковому відношенні. У випадку відсутності даних минулого місяця, в поле виводу динаміки зміни витрат виводиться повідомлення “Дані відсутні”.



Номер палива	Назва палива	Ціна палива	Витрата (одиниці)	Витрата (грн.)	Частка від загальних витрат (%)
113	Торф брикети	1700.0	2.2	3740.0	13.22
111	Вугілля брикет	1800.0	1.2	2160.0	7.63
117	Вуглеводневий газ	15000.0	0.8	12000.0	42.4
115	Топочний мазут	6500.0	1.6	10400.0	36.75

Рисунок 4.3.2 Детальне представлення параметризованого місячного звіту

Вікно представлення та обробки даних про палива (рисунок 4.3.3) має вигляд розділеної навпіл робочої області, ліву частину якої виділено на область табличного представлення даних про палива, що вже наявні в таблиці “Fuel” в базі даних, а саме номер-ідентифікатор палива, назва палива, та ціна за умовну одиницю палива. Права частина робочого простору являє собою інструментарій для додавання, видалення та редагування записів у таблиці, а також фільтрації вже наявних записів по певному параметру, який має бути введений користувачем у відповідне текстове поле

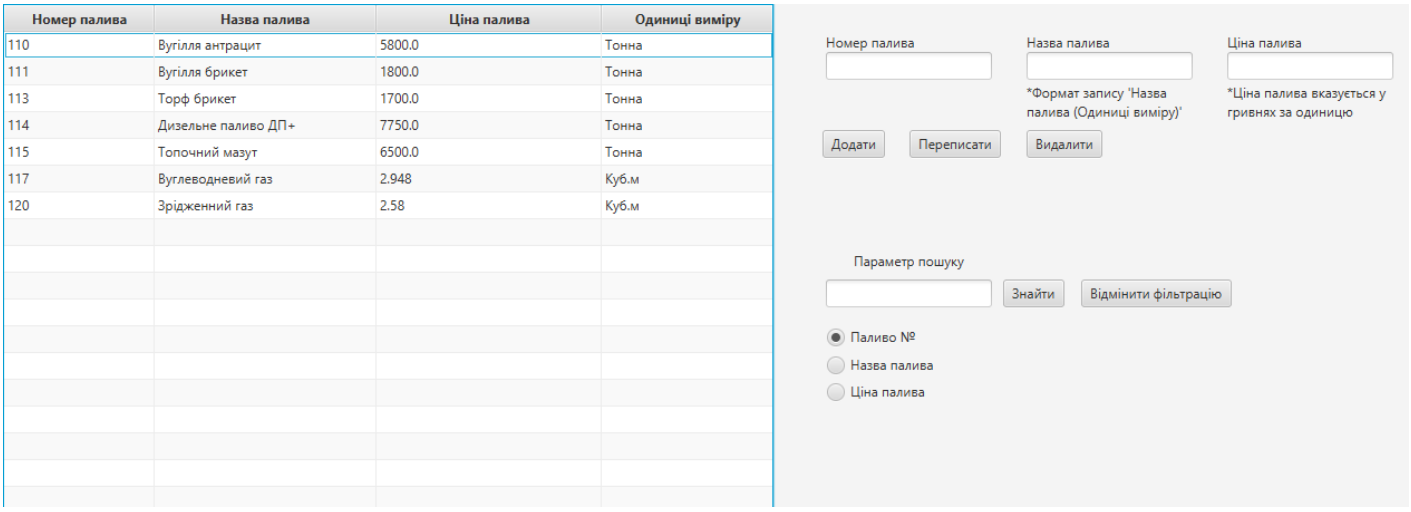


Рисунок 4.3.3 Користувацьке вікно обробки даних про паливо

Вікно представлення та обробки даних про витрати палива (рисунок 4.3.4) має структуру схожу на вікно обробки даних про палива, а також ідентичний функціонал для роботи із таблицею. Схожість побудови двох даних вікон інтерфейсу пояснюється ідентичністю задач, які вирішують дані вікна, а саме: додання нового запису в таблицю, редагування існуючого запису, видалення запису та фільтрація даних.

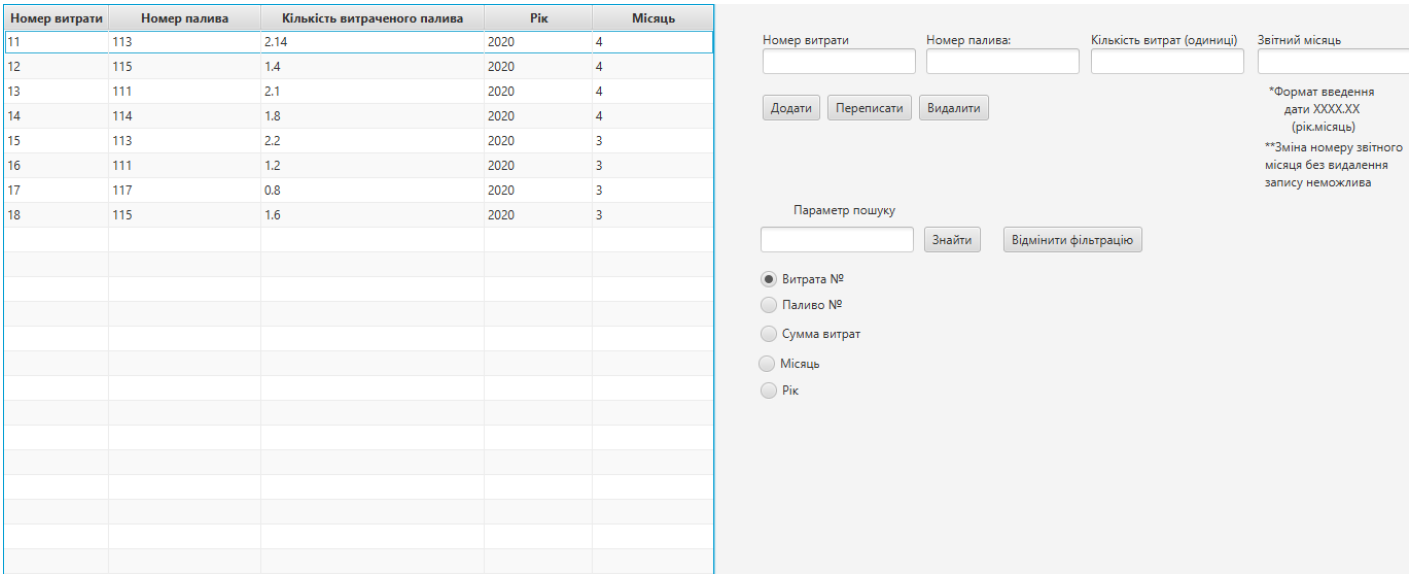


Рисунок 4.3.4 Користувацьке вікно обробки даних про витрати палив

4.4 Опис алгоритмів роботи програмної реалізації

Алгоритми взаємодії створюваного програмного комплексу із базою даних побудовані на основі драйверу JDBC, який забезпечує взаємодію Java-додатку із базою даних із використанням класів:

- “java.sql.Connection” – клас-конектор, що відкриває з’єднання із базою даних, яке в подальшому може використовуватися іншими модулями системи.
- “java.sql.Statement” – клас-обробник, який забезпечує виконання базою даних переданих їй SQL-запитів.
- “java.sql.ResultSet” – клас-контейнер, що являє собою результуючий набір даних, отриманих додатком від бази даних після виконання переданого SQL-запиту.

Інструкції до бази даних представлені класичними параметризованими SQL-запитами (SELECT, INSERT, UPDATE, DELETE), що виконуються за допомогою класу “java.sql.Statement” і оброблюються всередині програмної системи класами-виконавцями за посередством об’єкту класу “java.sql.ResultSet”.

Загальний функціонал створюваної системи представлений наступним набором користувацьких інструментів:

- Додавання запису в таблицю бази даних (рисунок 4.4.1), для чого створюється відповідний INSERT-запит, який передається на виконання базі даних після чого, новостворений запис додається у візуальну таблицю.
- Видалення обраного запису (рисунок 4.4.2) виконується з допомогою SQL-запиту DELETE, після виконання якого, обраний користувачем для видалення запис видаляється як із таблиці в базі даних, так й із візуального представлення цієї таблиці у вікні інтерфейсу користувача. В даному випадку було видалено запис номер 112.

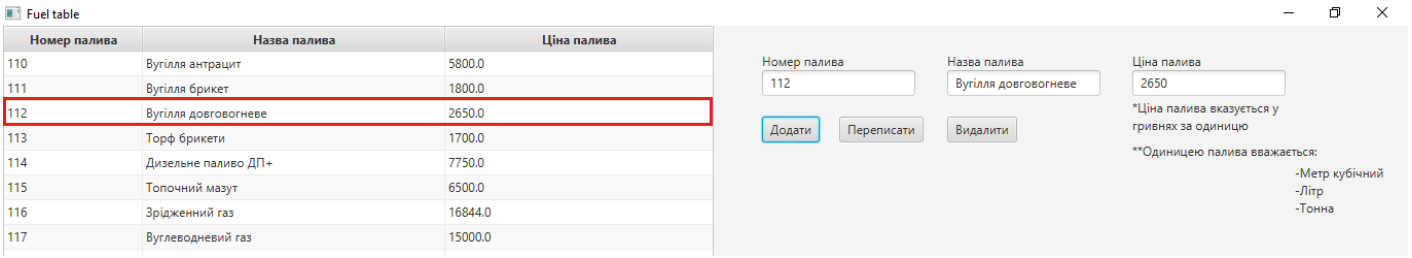


Рисунок 4.4.1 Створення нового запису в таблиці “Fuel”

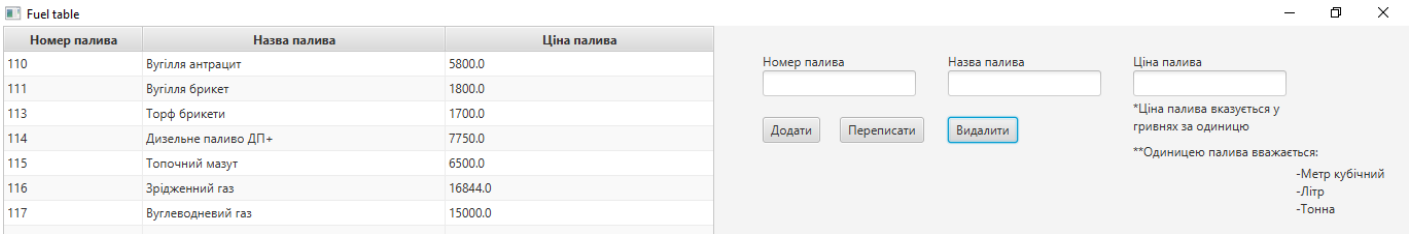


Рисунок 4.4.2 Видалення запису номер 112 із таблиці “Fuel”

- Переписання даних у записі (рисунок 4.4.3) забезпечується параметризованим SQL-запитом UPDATE, параметрам якого, значення передаються автоматично в залежності від того, які поля даних були заповнені користувачем. В даному випадку паливу “Зріджений газ” було переприсвоєне ідентифікаційний номер 120.

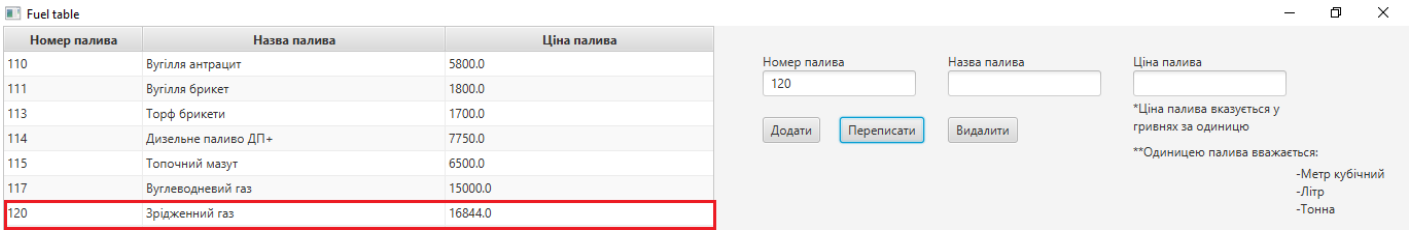


Рисунок 4.4.3 Переписання запису в таблиці “Fuel”

- Фільтрація записів по обраному полю (рисунок 4.4.4) виконується за допомогою SELECT-запиту, після виконання якого, програма, посередством об’єкту “java.sql.Statement” отримує об’єкт класу “java.sql.ResultSet”, який

піддається обробці алгоритмом фільтрації, після чого в візуальне представлення таблиці на інтерфейсному вікні користувача вносяться лише ті записи із цільової таблиці, які відповідають умові, яка була введена користувачем. В даному випадку було введено умову фільтрації по полю номера палива із параметром 114. Запис який відповідає даній умові було виведено в візуальне представлення таблиці.

Рисунок 4.4.4 Фільтрація записів із таблиці “Fuel”

- Сортування записів в візуальному представленні таблиці виконується вбудованим методом “sort()”, який належить класу javafx.scene.control.TableView, який, в свою чергу наслідується від класу Parent (рисунок 3.3.2).

  Сортування викликається через натискання на табличне поле, по якому необхідно виконувати сортування. Перше натискання на поле викличе сортування за зростанням (рисунок 4.4.5) або за алфавітом (для тектових значенн). Друге натискання викличе зворотньє сортування за зменшенням, або за зворотнім алфавітним порядком. Третє натискання, поверне сортування в нейтральний, тобто початковий стан.

Алгоритм вибору порядку сортування вказано з урахуванням нейтрального стану, як вихідного.

Номер палива	Назва палива	Ціна палива	Одиниці виміру
110	Вугілля антрацит	5800.0	Тонна
111	Вугілля брикет	1800.0	Тонна
113	Торф брикет	1700.0	Тонна
114	Дизельне паливо ДП+	7750.0	Тонна
115	Топочний мазут	6500.0	Тонна
117	Вуглеводневий газ	2.948	Куб.м
120	Зріджений газ	2.58	Куб.м

Рисунок 4.4.5 Сортування таблиці за зростанням по полю “Номер палива”

4.5 Висновки до розділу

В даному розділі було описано програмну реалізацію системи, представлені зображення, що демонструють роботи системи, а також описані основні алгоритми, які забезпечують основний функціонал програмної системи. Крім того продемонстровано роботу деяких функцій системи, та описані основні програмні компоненти.

Також була продемонстрована концептуальна модель база даних із якою взаємодіє система, описані зв’язки між таблицями цієї бази даних та продемонстрована композиція програмної системи, що в основі складається із класів-оброблювальників, класу-конектору, додаткової бібліотеки драйверу підключення до бази даних, декількох буферних (допоміжних класів) та папки із конфігураційними файлами інтерфейсу у форматі “.xml”, які виконують роль основних інтерфейсних файлів.

5 МОДЕЛЬ ВЗАЄМОДІЇ КОРИСТУВАЧА ІЗ ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Системні вимоги до інсталяції та розгортання програмної системи

Мінімальні системні вимоги до комп'ютера, на якому буде розгорнутий сервер із базою даних визначені як:

- Операційна система Microsoft Windows XP, Microsoft Windows 7, Microsoft Windows 10;
- Мінімально 512 Мбайт ОЗУ (рекомендовано 1024 Мбайт і більше);
- 1024 Мбайт вільного місця на диску для розгортання серверу;
- Процесор: Intel Pentium 2 (226 МГц);
- Архітектура процесора: 32-bit;

Мінімальні системні вимоги до комп'ютера користувача за умови віддаленого підключення до серверу (у випадку розгортання серверу та запуску программи на одному комп'ютері, встановлюються вищеописані системні вимоги до комп'ютера для розгортання серверу):

- Операційна система Microsoft Windows 7, Microsoft Windows 10
- Мінімально 128 Мбайт ОЗУ (рекомендовано 512 Мбайт і більше)
- Простір на диску: 124 Мбайт для JRE, 2 Мбайт для оновлення Java
- Процессор: Intel Pentium 2 (266 МГц)

Проте, не зважаючи на низькі системні вимоги до встановлення сервера бази даних та середовища виконання Java, для забезпечення більш високої швидкості виконання робочих операцій, а, відповідно, для більш комфортної та ефективної взаємодії

користувача із системою, рекомендовано встановлення якомога більших системних потужностей, як на комп'ютері користувача, так і на комп'ютері, сервері.

5.2 Інсталяція і запуск системи

Так як розроблюваний програмний продукт працює із реляційною базою даних, що встановлена на віртуальному сервері, в першу чергу необхідно встановити програмне забезпечення для підключення серверу (MySQL Server), та створення нової бази даних на ньому.

Середовище MySQL в тому числі програма MySQL Server надається компанією Oracle в режимі вільного доступу через офіційні веб-сайти тому, копія програми MySQL Installer Community Edition версії 8.0.11.0 додано до основної збірки створюваного програмного продукту.

Для встановлення серверу бази даних MySQL Server необхідно запустити файл (рисунок 5.2.1) "mysql-installer-community-8.0.11.0" в папці, яка була створена після розпакування архіву із збіркою програмного продукту

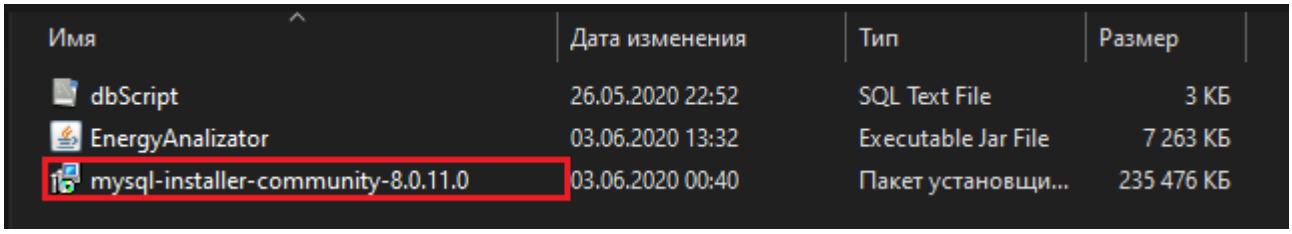


Рисунок 5.2.1 Файл встановлення середовища MySQL

Після запуску програми – встановлювача необхідно буде підтвердити згоду із ліцензією "General Public License" (рисунок 5.2.2), що надає користувачу право вільно копіювати, модифікувати, використовувати та поширювати (в тому числі на комерційних засадах) програмне забезпечення. GPL надає права вільного запуску програмного забезпечення із будь-якою ціллю, свободу вільного вивчення алгоритмів

роботи програми (із умовою попереднього доступу до вихідного коду), свободу вільного модифікування та розповсюдження у вільному доступі як самої програми, так і модифікацій до неї (також за умови попереднього доступу до вихідного коду).

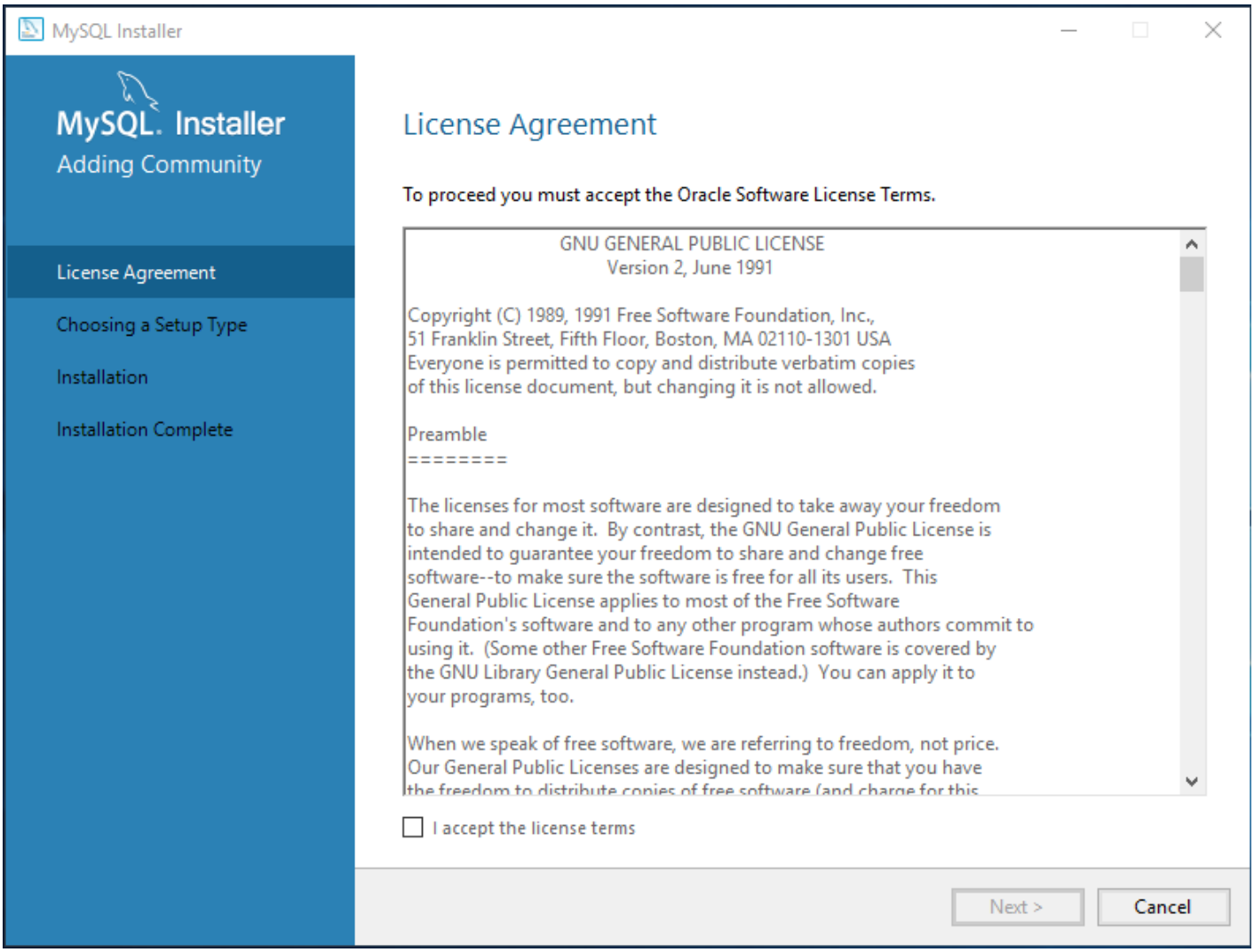


Рисунок 5.2.2 Підтвердження згоди із ліцензією GNU GPL

У вікні вибору набору встановлюваних програм із середовища MySQL (рисунок 5.2.3) необхідно вибрати користувацький варіант (Custom Install), який надає можливість самостійно обрати функціонал для встановлення, який необхідно для виконання даної конкретної задачі, через що навантаження на операційну систему та апаратну частину комп’ютера приводиться до оптимізованих значень.

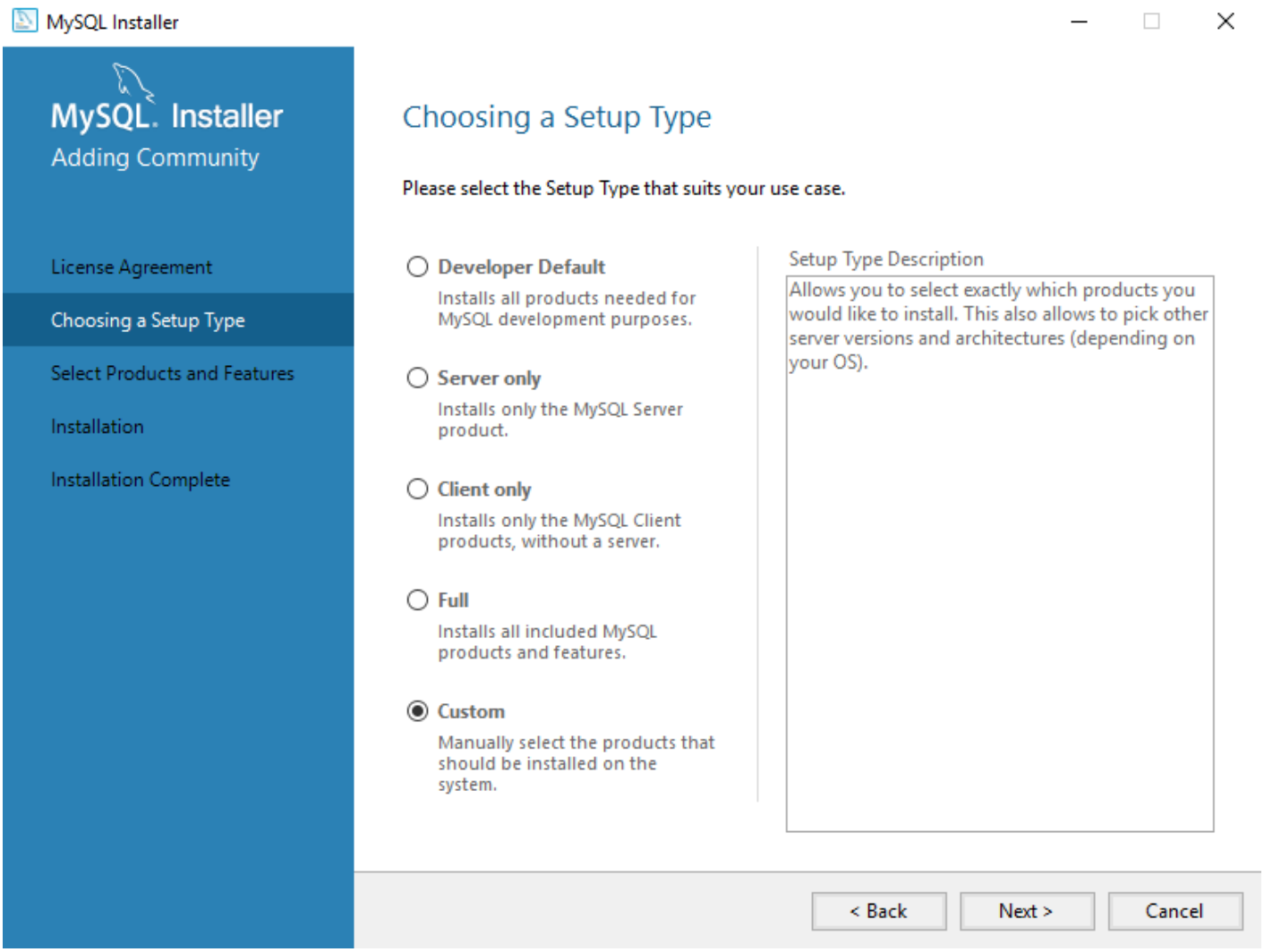


Рисунок 5.2.3 Вікно вибору типу встановлення

На рисунку 5.2.4 представлено список програм, які необхідно обрати для отримання необхідного функціоналу середовища бази даних MySQL, який використовується програмною системою.

Мінімальний необхідний функціонал середовища MySQL складається із трьох програм:

- MySQL Server 8.0.11 – X64
- MySQL Workbench 8.0.11 – X64
- Connector/J 8.0.11 – X64 (Connector JDBC)

Для вибору даних програм в інсталері необхідно розгорнути до кінця дерева “MySQL Server”, “Applications” та “MySQL Connectors/Connector/J” після чого перенести кінцеві збірки даних програм в праве меню через вибір програми та натискання зеленої стрілки в інтерфейсі вікна. Після виконання даних дій необхідно натиснути кнопку “Next”.

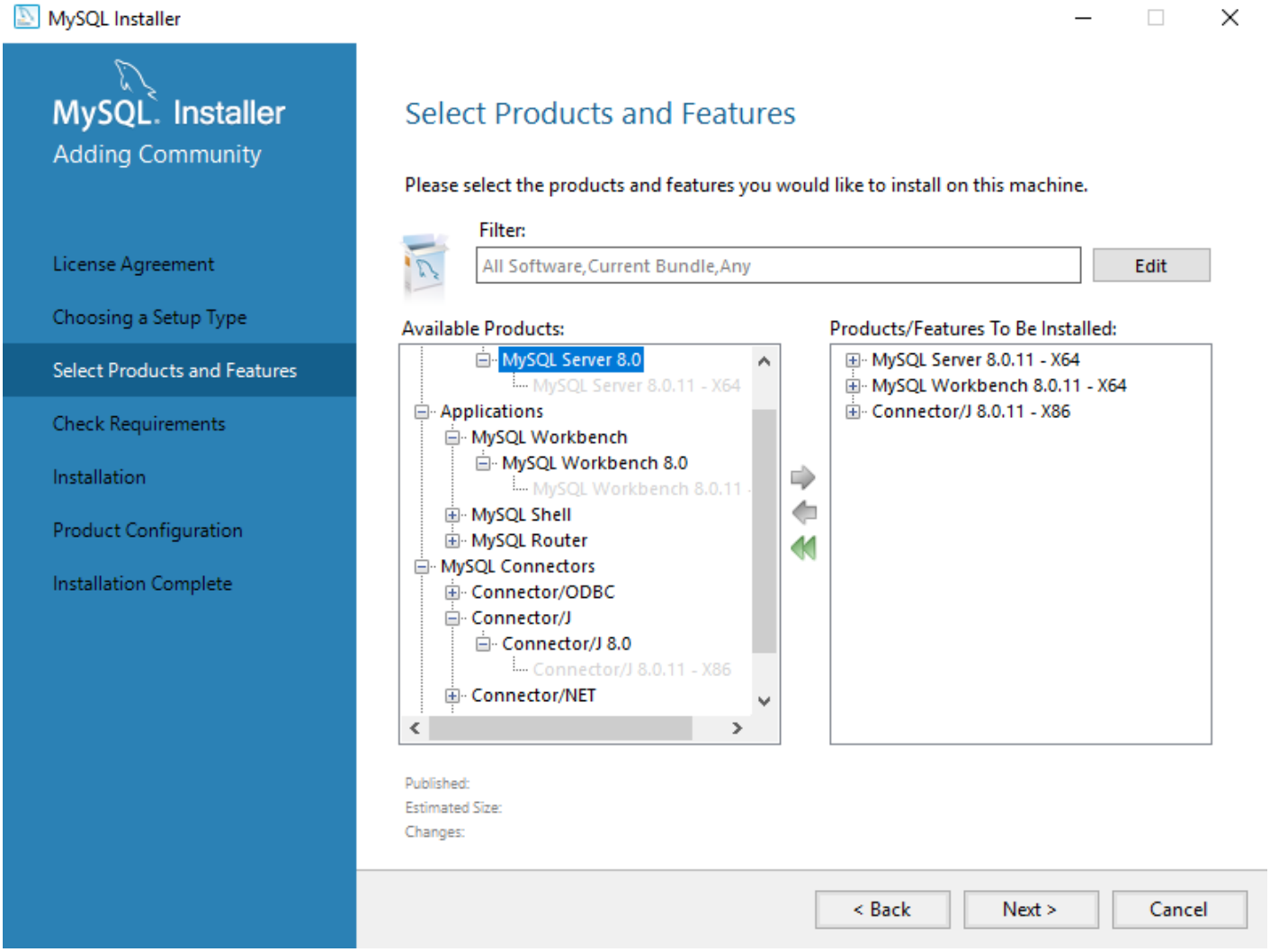


Рисунок 5.2.4 Вибір програмних компонентів для встановлення

В наступному вікні (рисунок 5.2.5) необхідно натиснути кнопку “Execute” та дочекатись завершення передвстановлення MySQL Server та MySQL Workbench. По завершенні процесу передвстановлення обидва маркери перейдуть у стан готовності.

Після цього натискання кнопки “Next”, направить користувача безпосередньо до процесу встановлення програм на комп’ютер.

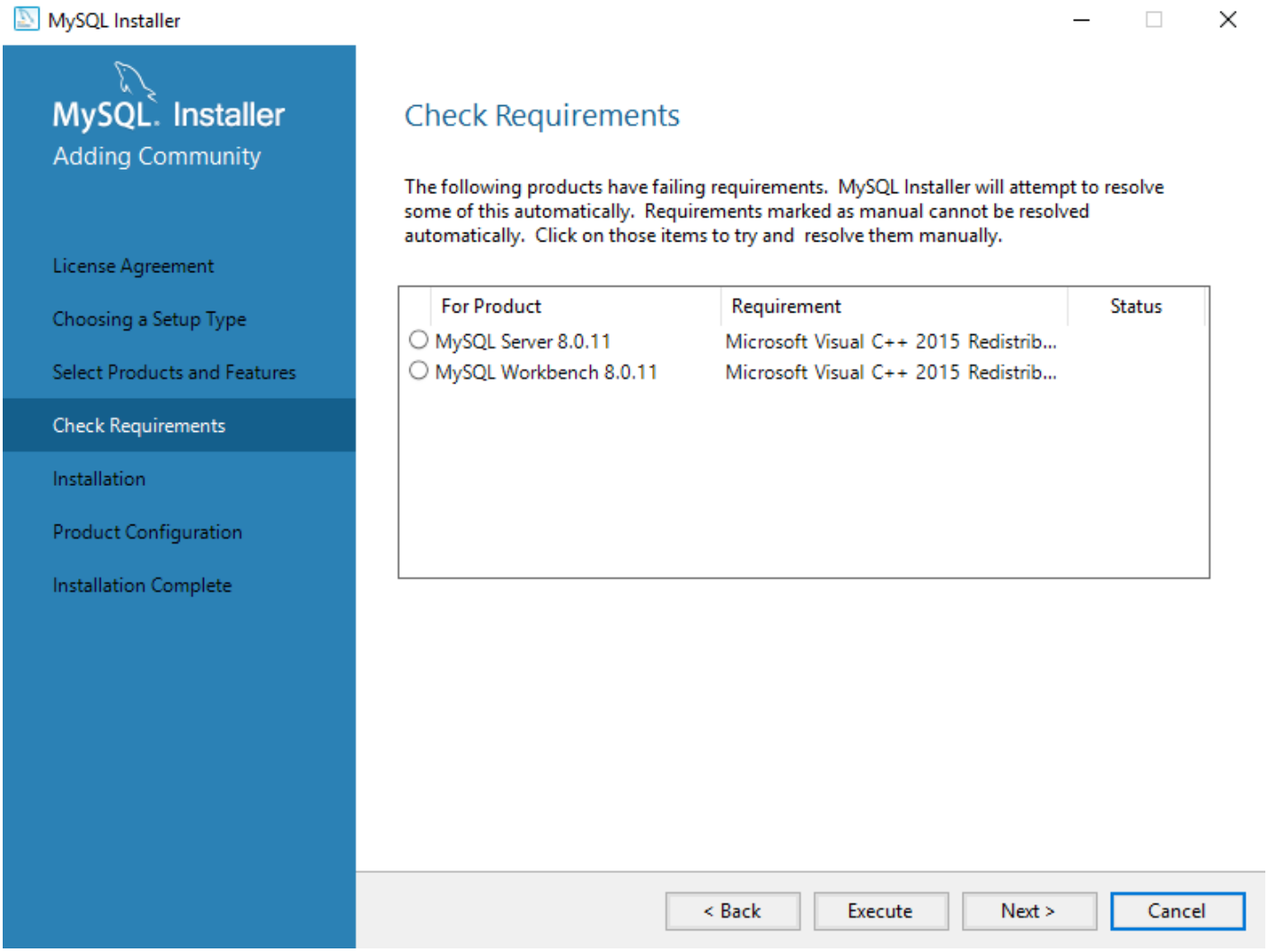


Рисунок 5.2.5 Підтвердження встановлення програм оточення

У вікні безпосередньої інсталяції програмних засобів (рисунок 5.2.6) необхідно натиснути кнопку “Execute” і дочекатися встановлення MySQL Server, MySQL Workbench та коннектора Connector/J (JDBC).

Після завершення встановлення даних програмних засобів і натискання кнопки “Next” в інтерфейсі інсталятора, буде виконане налаштування MySQL Server.

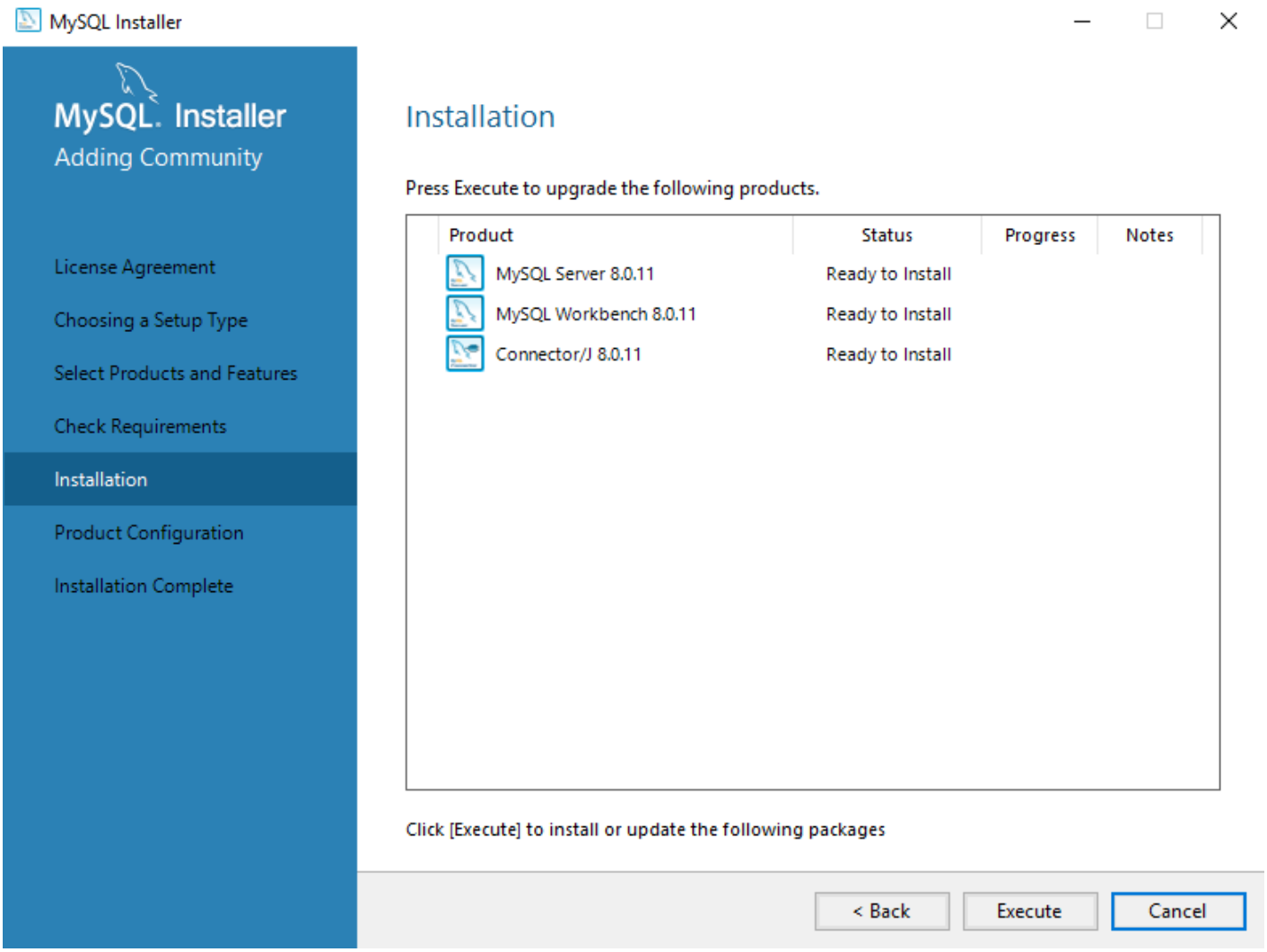


Рисунок 5.2.6. Вікно інсталяції програмних засобів

На рисунку 5.2.7 представлено вікно налаштування облікового запису адміністратора у MySQL Server. Тут можна налаштувати пароль облікового запису адміністратора, а також додати нові користувацькі облікові записи та налаштувати права доступу до даних, якими будуть володіти користувачі. Проте для мінімального функціоналу необхідно лише встановити пароль, підтвердити його у відповідних текстових полях та перейти далі натисканням кнопки “Next”. В даному випадку для облікового запису адміністратора (акаунта суперкористувача) було використано класичний пароль “root”.

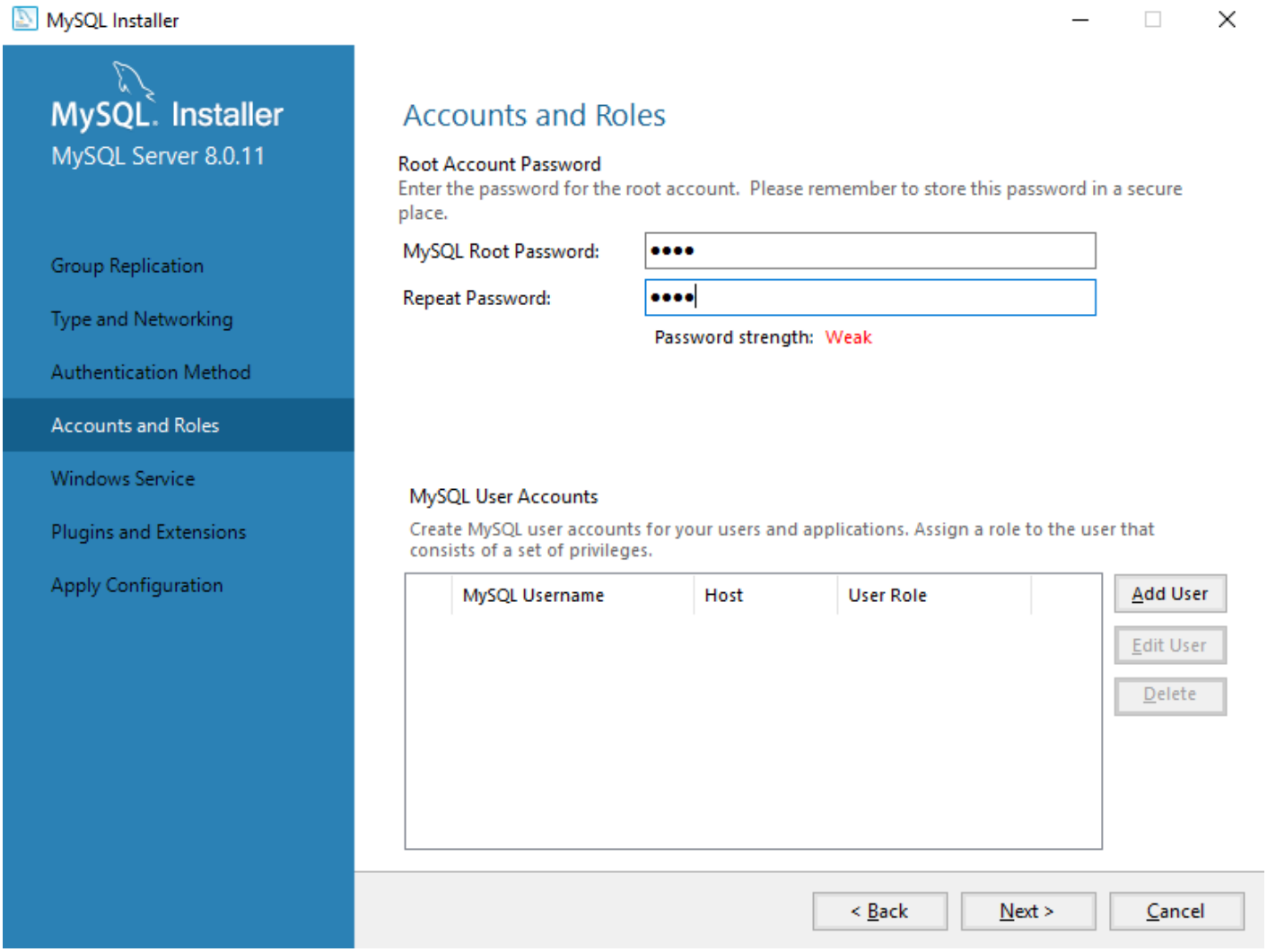


Рисунок 5.2.7 Вікно налаштування облікових записів MySQL Server

На рисунку 5.2.8 представлено вікно підтвердження налаштувань, де, після натискання кнопки “Execute” буде запущений процес конфігурування робочого середовища. Етап конфігурування MySQL Server:

- Редагування та запис конфігураційного файлу;
- Додання MySQL Server у список виключень для Windows Firewall;
- Налаштування служб Windows для роботи із MySQL Serve;
- Ініціалізація бази даних;
- Запуск сервера;
- Підтвердження налаштувань безпеки;

- Створення визначених вище користувацьких облікових записів;

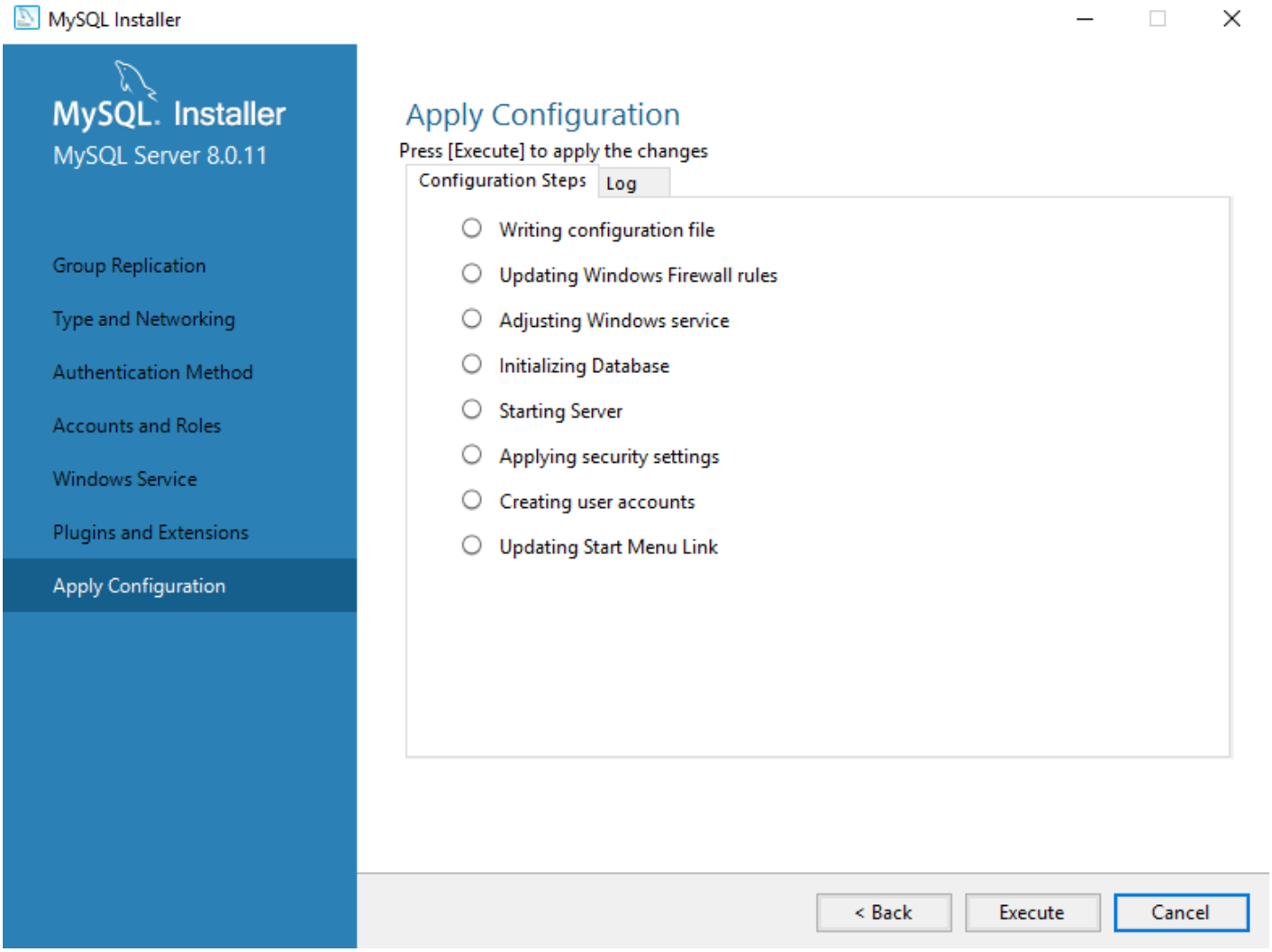


Рисунок 5.2.8 Вікно підтвердження конфігурації серверу

Після завершення процесу конфігурування та відкриття вікна завершення налаштування та інсталяції (рисунок 5.2.9), в якому можна зберегти дані про відслідковування процесу конфігурування, а також обрати чи буде відкрите середовище MySQL Workbench одразу після закриття даного вікна, середовище MySQL вважається встановленим, налаштованим та готовим до використання.

Для початку роботи із програмною системою необхідно відкрити MySQL Workbench, створити в ньому нове підключення із стандартними параметрами до

серверу MySQL Server, та встановити на ньому базу даних, створену для даної програмної системи.

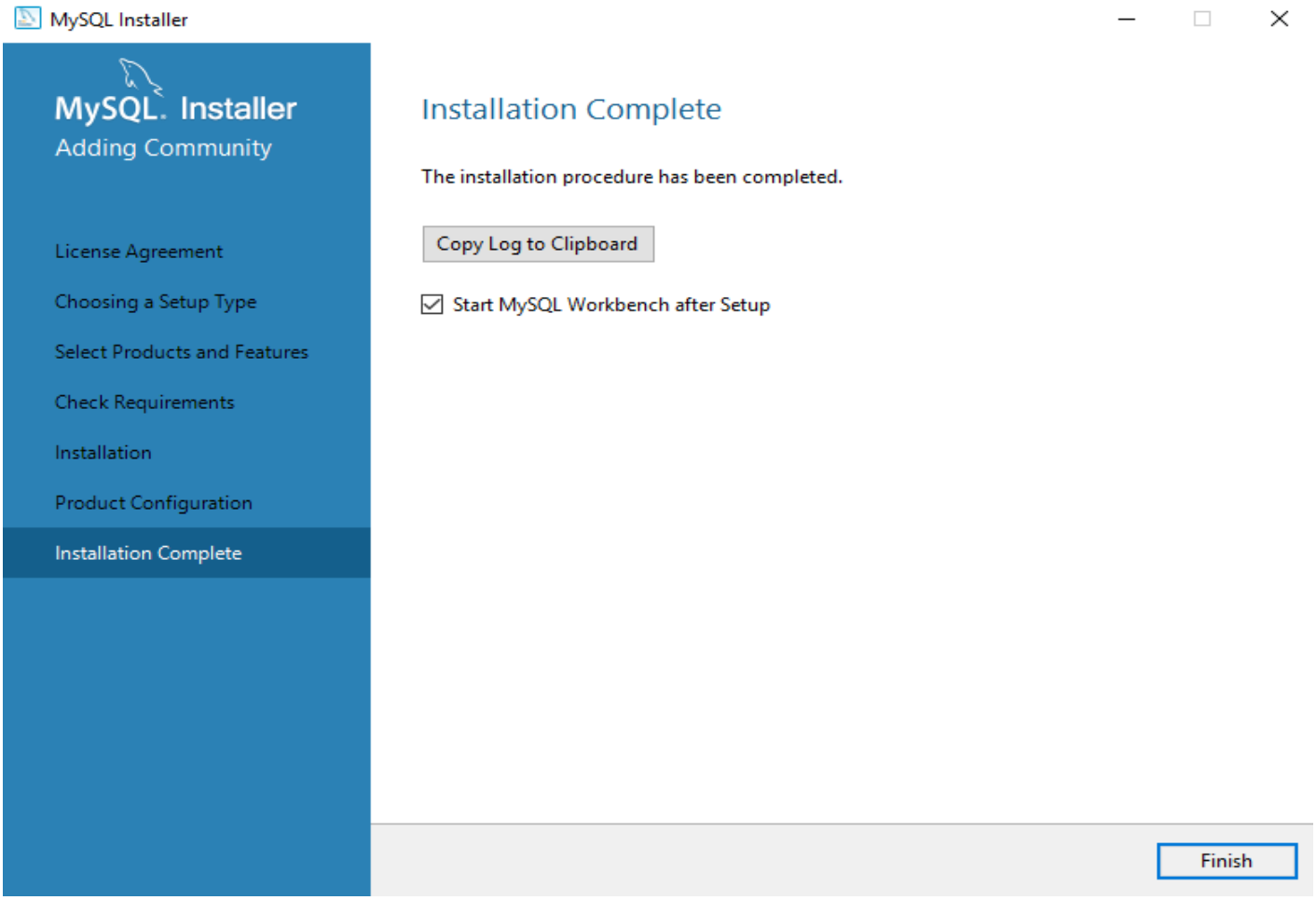


Рисунок 5.2.9 Вікно завершення інсталяції середовища

Створення бази даних на встановленому сервері виконується за рахунок виконання у інтерфейсі MySQL Workbench конфігураційного файлу бази даних (рисунок 5.2.10) “dbScript” із розширенням “SQL Text File”.

Після обробки файлу засобами середовища та встановлення бази даних на сервер, середовище MySQL готове до роботи і може використовуватись програмною системою.

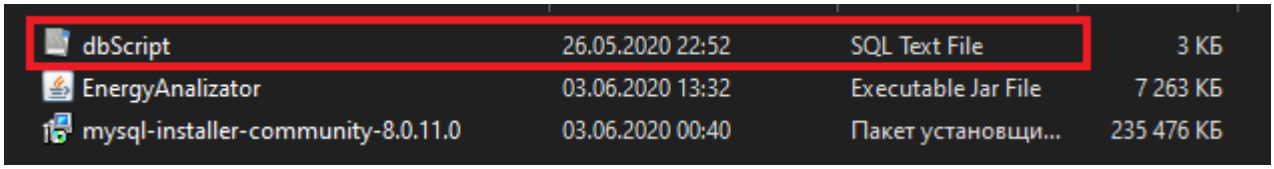


Рисунок 5.2.10 Конфігураційний файл бази даних

Запуск програмної системи виконується через відкриття файлу “EnergyAnalizator” в папці із збіркою системи. Після завантаження та підключення системи до серверу бази даних, буде відкрите початкове вікно користувацького інтерфейсу (рисунок 4.3.1), де користувач отримує доступ до всіх можливостей програмної системи, тобто може:

- Створювати нові звіти на основі попередньо створених записів про витрати;
- Переглядати поточні звіти із можливістю перегляду деталізованої інформації по обраному звіту;
- Додавати, редагувати та видаляти записи про використовувані палива;
- Додавати, редагувати та видаляти записи про витрати кожного конкретного палива в кожному конкретному місяці;
- Переглядати таблиці із інформацією про палива та витрати палив в кожному місяці;
- Проводити фільтрацію записів в кожній з таблиць задля полегшення навігації по таблиці, а також спрощення аналізу наявних даних;
- Виставляти записи в кожній із таблиць в пріоритетному порядку за збільшенням, зменшенням, в алфавітному чи інвертованому алфавітному порядку;
- Отримувати діагностичні повідомлення від системи, про некоректність введених даних або інші помилки користувацькі помилки, припущені при роботі із системою;

Після завершення роботи із системою, для коректного завершення робочого сеансу, необхідно закрити всі відкриті вікна системи. За необхідністю, можливо зупинити сервер, викликавши MySQL Notifier із панелі задач, викликавши на ньому контекстне меню та виконавши в ньому команду “Stop”. Після того як сервер перейде із стану “Running” в стан “Stopped” через проміжний стан “StopPending”, сервер зупинено, робота із системою завершена. Для повторного запуску сервера необхідно знову викликати MySQL Notifier та виконати в ньому команду “Start”, що також знаходиться

в контекстному меню. Після завершення запуску, сервер перейде в стан “Running” через проміжний стан “StartPending” і буде готовим до взаємодії.

Для відновлення з’єднання із базою даних на знову підключеному сервері необхідно відкрити середовище MySQL Workbench, активувати з’єднання (рисунок 5.2.11) та ввести пароль адміністратора або користувача для даного підключення. Після завершення завантаження, база даних готова до роботи із програмною системою.

Рисунок 5.2.11 Відновлення з’єднання із базою даних

Для коректного завершення роботи із програмною системою необхідно закрити всі відкриті вікна користувацького інтерфейсу і дочекатися завершення роботи програмної системи. Збереження змін у базі даних та закриття з’єднання із сервером проводяться алгоритмами програмної системи автоматично та не потребують втручання користувача.

5.3 Висновки до розділу

В даному розділі були презентовані дії користувача для початку роботи із системою, також були описані системні вимоги як для ситуацій розгортання серверу бази даних та програмної системи на одному комп’ютері, так і для ситуацій розгортання серверу та запуску програмної системи на різних комп’ютерах. Крім того були описані необхідні дії для встановлення та конфігурування мінімально необхідного набору програм, сервісів та драйверів для роботи із створеною програмною системою.

Сама система надає користувачу такі можливості:

- Створення нових та перегляд вже існуючих місячних звітів із узагальненою інформацією;
- Перегляд деталізацій використання кожного конкретного палива в кожному конкретному місяці, побудованих на основі обраного користувачем звіту із списку існуючих;
- Адміністрування таблиці із інформацією про використовувані палива;
- Адміністрування таблиці із даними про використання палив в кожному місяці;
- Виставлення фільтру вибору записів в кожній із таблиць в ручному режимі, що дозволяє більш ефективно користуватись системою;
- Використання системи виставлення пріоритетного відображення даних в кожній із таблиць, що значно спрощує навігацію по таблицям;

ВИСНОВКИ

В ході виконання дипломної роботи було виконано дослідження на тему обліку використання енергоресурсів на об’єктах виробництва, була проаналізована предметна область та існуючі системи обліку енергоресурсів. Було розглянуте поняття автоматизованої системи комерційного обліку енергоресурсів та досліджена система “АСКОЕ Пульсар”.

Під час дослідження було виявлено, що “АСКОЕ Пульсар” є достатньо об’ємною, високо-автоматичною та автономною системою, яка може виконувати функції контролю, обліку та регулювання використання енергоресурсів. Проте, об’ємність даної системи спричиняє собою проблему високої вартості як самої системи, так і кожного апаратного модулю та її технічного обслуговування, більш того для обслуговування апаратної частини системи необхідні спеціалісти високої кваліфікації.

Результатом виконання дипломної роботи стала програмна система, призначена для ведення обліку використання енергоресурсів на об’єктах виробництва. Система складається із інтерфейсу користувача, візуального представлення даних системи в інтерфейсі, алгоритмів обробки даних, та бази даних яка, власне, і зберігає всю інформацію, якою оперує система.

Розроблений додаток надає наступний функціонал:

- Створення та перегляд узагальнених та деталізованих про використання енергоресурсів;
- Внесення, зберігання та редагування інформації про витрати палив, що використовуються на виробництві;
- Відслідковування динаміки зміни щомісячних фінансових витрат на закупівлю палив;
- Перегляд усієї доступної інформації у простому табличному вигляді;

- Налаштування режиму перегляду доступної інформації завдяки функціоналу фільтрування та впорядкування записів в таблицях;

Створена система має наступні переваги перед доступними аналогами:

- Зрозумілість – програмна система має простий і зрозумілий інтерфейс, а також передбачені модулі відслідковування помилок вводу даних із виводом діагностичних повідомлень, які надають інформацію про помічену помилку;
- Модифіковність – програмна система має відкритий вихідний код та може використовуватись, модифікуватись та доповнюватись у вільному порядку;
- Безкоштовність – програмна система виконана та працює із програмними засобами, які розповсюджуються видавцями у вільному доступі тому дана програма не потребує купівлі високовартісних програмних засобів для своєї роботи;
- Універсальність – створена програмна система може бути адаптована під більшість виробництв, де використовуються енергоматеріали в якості палива через те, що база даних системи побудована таким чином, що може приймати та зберігати інформацію про більшість доступних палив.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Бенджамин Дж. Эванс, Джеймс Гоф, Крис Ньюленд. Java: оптимизация программ. Практические методы повышения производительности приложений в JVM. — М.: Диалектика, 2019. — 448 с.
2. Герберт Шилдт. Java. Полное руководство, 10-е. — М.: Диалектика, 2018. — 1488 с.
3. К. Дж. Дейт SQL и реляционная теория. Как грамотно писать код на SQL. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 480 с.
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368 с.
5. Офіційна документація СУБД “MySQL” [Електронний ресурс] – режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.0/en/
6. Офіційна документація мови програмування Java [Електронний ресурс] – режим доступу до ресурсу: https://docs.oracle.com/en/java/index.html.
7. Офіційна документація драйверу JDBC API [Електронний ресурс] – режим доступу до ресурсу: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
8. Офіційна документація проекту Apache Maven Project [Електронний ресурс] – режим доступу до ресурсу: https://maven.apache.org/guides/ .
9. Офіційна документація середовища Intelij Idea [Електронний ресурс] – режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/idea/documentation/ .
10. Офіційна документація платформи JavaFX [Електронний ресурс] – режим доступу до ресурсу: https://docs.oracle.com/javase/8/javafx/api/toc.htm.

Додаток 1

Розробка програмного забезпечення для обліку використання енергоресурсів на об’єктах виробництва на платформі ОС Windows

Специфікація

УКР.НТУУ“КПІ”.ТР61117_20Б

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ “КПІ”. ТР61117_20Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ “КПІ”. ТР61117_20Б 12-1	Текст програмного модулю	
УКР.НТУУ “КПІ”. ТР61117_20Б 13-1	Опис програми	

Додаток 2

Розробка програмного забезпечення для обліку використання енергоресурсів на об’єктах виробництва на платформі ОС Windows

Текст програмного модулю

УКР.НТУУ“КПІ”.ТР61117_20Б 12-1

Аркушів 10

2020

```
/*Class ReportCreator*/
package creators;

import bufferClasses.SpendOfFuel;
import connectors.Connector;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import tables.Expense;
import tables.Report;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

public class ReportCreator {

    @FXML
    TableView<Report> table;
    @FXML
    TableColumn <Report, Integer> reportIdFXML;
    @FXML
    TableColumn <Report, Integer> yearFXML;
    @FXML
    TableColumn <Report, Integer> monthFXML;
    @FXML
    TableColumn <Report, Double> expenseSumFXML;

    private ObservableList<Report> report = FXCollections.observableArrayList();
    private Connection connection;
    private ResultSet resultSet;
    private Statement statement = null;
    @FXML
    private void initialize () {
        table.getSelectionModel().setSelectionMode(
                SelectionMode.MULTIPLE
        );
        report.clear();
        connection = Connector.CONNECTOR.getConnection();
        try {
            statement = connection.createStatement();
            resultSet = statement.executeQuery("select * from report");
            int i = 1;
            while (resultSet.next()) {
                report.add(new Report(resultSet.getInt(2), (int)(resultSet.getInt(2) / 100),
resultSet.getInt(2) % 100, Double.parseDouble(resultSet.getString(1))));
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        reportIdFXML.setCellValueFactory(new PropertyValueFactory<Report,
Integer>("reportId"));
        yearFXML.setCellValueFactory(new PropertyValueFactory<Report, Integer>("year"));
        monthFXML.setCellValueFactory(new PropertyValueFactory<Report, Integer>("month"));
        expenseSumFXML.setCellValueFactory(new PropertyValueFactory<Report,
Double>("expenseSum"));
        table.setItems(report);
    }

    @FXML
    TextField reportDateTFXML;

    @FXML
    private void create () {
        ArrayList <SpendOfFuel> spendOfFuels = new ArrayList<SpendOfFuel>();
        int reportId = 0;
        try {
            reportId = Integer.parseInt(reportDateTFXML.getText().substring(0, 4) +
(reportDateTFXML.getText() + " ").substring(5, 7));
        } catch (NumberFormatException e) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Заповніть поле дати звіту");
            alert.showAndWait();
            return;
        }
        double expensesSum = 0;
        try {
            resultSet = statement.executeQuery("select fuelId, amountOfSpend from expense where
reportId = " + reportId);

            while (resultSet.next()) {
                spendOfFuels.add(new SpendOfFuel(resultSet.getInt(1),
Double.parseDouble(resultSet.getString(2))));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        for (int i = 0; i < spendOfFuels.size(); i++) {
            try {
                resultSet = statement.executeQuery("select fuelCost from fuel where fuelId = "
+ spendOfFuels.get(i).getFuelId());
                while (resultSet.next()) {

spendOfFuels.get(i).setFuelCost(Double.parseDouble(resultSet.getString(1)));
                }
                expensesSum += (spendOfFuels.get(i).getFuelCost() *
spendOfFuels.get(i).getAmountOfSpend());
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
        try {
            statement.executeUpdate("insert into report (reportId, expensesSum) values (" +
reportId + ", " + expensesSum + ")");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        initialize();
    }

    @FXML
    private void delete () {
        ObservableList <Report> cells = FXCollections.observableArrayList();
        cells = table.getSelectionModel().getSelectedItems();
        if (cells.size() == 0) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Оберіть хоча б один запис");
            alert.showAndWait();
        }
        for (int i = 0; i < cells.size(); i++) {
            int reportId = cells.get(i).getReportId();
            try {
                statement.executeUpdate("delete from report where reportId = " + reportId);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        initialize();
    }

    @FXML
    RadioButton reportIdRB;
    @FXML
    RadioButton expensesSumRB;
    @FXML
    RadioButton monthRB;
    @FXML
    RadioButton yearRB;
    @FXML
    TextField filter;

    @FXML
    public void filter(ActionEvent event) {
        String query = "select reportId, expensesSum from report where ";
        try {
            if (reportIdRB.isSelected()) {
                query += "reportId = " + Integer.parseInt(filter.getText());
            }

            if (expensesSumRB.isSelected()) {
                query += "expensesSum = " + Double.parseDouble(filter.getText());
            }

            if (monthRB.isSelected()) {
                query += "reportId % 100 = " + Integer.parseInt(filter.getText());
            }
```

```
            if (yearRB.isSelected()) {
                query += "truncate(reportId / 100, 0) = " + Integer.parseInt(filter.getText());
            }
        } catch (NumberFormatException e) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Заповніть поле параметру пошуку");
            alert.showAndWait();
            return;
        }

        doFilter(query);
    }

    public void doFilter (String query) {
        report.clear();
        table.getItems().clear();
        connection = Connector.CONNECTOR.getConnection();
        try {
            statement = connection.createStatement();
            resultSet = statement.executeQuery(query);
            int i = 1;
            while (resultSet.next()) {
                report.add(new Report(resultSet.getInt(1), (int)(resultSet.getInt(1) / 100),
resultSet.getInt(1) % 100, Double.parseDouble(resultSet.getString(2))));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        reportIdFXML.setCellValueFactory(new PropertyValueFactory<Report,
Integer>("reportId"));
        yearFXML.setCellValueFactory(new PropertyValueFactory<Report, Integer>("year"));
        monthFXML.setCellValueFactory(new PropertyValueFactory<Report, Integer>("month"));
        expenseSumFXML.setCellValueFactory(new PropertyValueFactory<Report,
Double>("expenseSum"));
        table.setItems(report);
    }
    @FXML
    private void goToFuel () {
        Parent root = null;
        try {
            root = FXMLLoader.load(getClass().getResource("/FXML/fuelWindow.fxml"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        Stage stage = new Stage();
        stage.setTitle("Fuel table");
        stage.setScene(new Scene(root, 1400, 700));
        stage.setMaximized(true);
        stage.show();
    }

    @FXML
    private void goToExpense () {
        Parent root = null;
```

```
        try {
            root = FXMLLoader.load(getClass().getResource("/FXML/expenseWindow.fxml"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        Stage stage = new Stage();
        stage.setTitle("Expenses table");
        stage.setScene(new Scene(root, 1400, 700));
        stage.setMaximized(true);
        stage.show();
    }

    private static Report selectedReport;
    public static Report getSelected () {
        return selectedReport;
    }

    @FXML
    private void moreDetails () {
        ObservableList <Report> cells = FXCollections.observableArrayList();
        cells = table.getSelectionModel().getSelectedItems();
        if (cells.size() == 0) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Оберіть хоча б один запис");
            alert.showAndWait();
            return;
        }
        Report selReport = cells.get(0);
        selectedReport = selReport;
        Parent root = null;
        try {
            root = FXMLLoader.load(getClass().getResource("/FXML/detailsReportWindow.fxml"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        Stage stage = new Stage();
        stage.setTitle("Report details");
        stage.setScene(new Scene(root, 1400, 700));
        stage.setMaximized(true);
        stage.show();
    }
}

/*Class ExpenseCreator*/
public class ExpenseCreator {

    @FXML
    TableView <Expense> table;
    @FXML
    TableColumn<Expense, Integer> expenseIdFXML;
    @FXML
    TableColumn<Expense, Integer> fuelIdFXML;
    @FXML
    TableColumn<Expense, Double> amountOfSpendFXML;
    @FXML
```

```
    TableColumn<Expense, Integer> yearFXML;
    @FXML
    TableColumn<Expense, Integer> monthFXML;

    @FXML
    TextField fuelIdTFXML;
    @FXML
    TextField amountOfSpendTFXML;
    @FXML
    TextField yearMonthTFXML;
    @FXML
    TextField filter;
    @FXML
    RadioButton expenseId;

    private ObservableList<Expense> expense = FXCollections.observableArrayList();
    private Connection connection;
    private ResultSet resultSet;
    private Statement statement;
    @FXML
    private void initialize () {
        table.getSelectionModel().setSelectionMode(
                SelectionMode.MULTIPLE
        );
        expense.clear();
        connection = Connector.CONNECTOR.getConnection();
        try {
            statement = connection.createStatement();
            resultSet = statement.executeQuery("select expenseId, fuelId, amountOfSpend, year,
month from expense");
            int i = 1;
            while (resultSet.next()) {
                expense.add(new Expense(resultSet.getInt(1), resultSet.getInt(2),
resultSet.getInt(4), resultSet.getInt(5), Double.parseDouble(resultSet.getString(3))));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        expenseIdFXML.setCellValueFactory(new PropertyValueFactory<Expense,
Integer>("expenseId"));
        fuelIdFXML.setCellValueFactory(new PropertyValueFactory<Expense, Integer>("fuelId"));
        amountOfSpendFXML.setCellValueFactory(new PropertyValueFactory<Expense,
Double>("amountOfSpend"));
        yearFXML.setCellValueFactory(new PropertyValueFactory<Expense, Integer>("year"));
        monthFXML.setCellValueFactory(new PropertyValueFactory<Expense, Integer>("month"));
        table.setItems(expense);
    }

    @FXML
    public void insert(ActionEvent event) {
        int expenseId = 0;
        int fuelId = 0;
        double amountOfSpend = 0.0;
        int year = 0;
        int month = 0;
        try {
            fuelId = Integer.parseInt(fuelIdTFXML.getText());
```

```
            expenseId = Integer.parseInt(expenseIdTFXML.getText());
            amountOfSpend = Double.parseDouble(amountOfSpendTFXML.getText());
            year = Integer.parseInt(yearMonthTFXML.getText().substring(0, 4));
            month = Integer.parseInt((yearMonthTFXML.getText() + " ").substring(5, 7));
        } catch (NumberFormatException e) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Всі поля повинні бути заповнені");
            alert.showAndWait();
            return;
        }
        try {
            statement.executeUpdate("insert into expense (expenseId, fuelId, year, month,
amountOfSpend, reportId) values (" + expenseId + ", "+ fuelId + ", " + year + ", " + month + ",
" + amountOfSpend +", " + (year * 100 + month) + ")");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        initialize();
    }

    @FXML
    TextField expenseIdTFXML;

    @FXML
    public void upate(ActionEvent event) {
        int idB = 0;
        int expenseId = 0;
        int fuelId = 0;
        double amountOfSpend = 0.0;
        int year = 0;
        int month = 0;
        int reportId = 0;
        int i = 0;
        try {
            expenseId = Integer.parseInt(expenseIdTFXML.getText());
            i++;
        } catch (NumberFormatException e) {
        }

        try {
            month = Integer.parseInt((yearMonthTFXML.getText() + " ").substring(5, 7));
            year = Integer.parseInt(yearMonthTFXML.getText().substring(0, 4));
            i++;
        } catch (StringIndexOutOfBoundsException e) {
        }

        try {
            amountOfSpend = Double.parseDouble(amountOfSpendTFXML.getText());
            i++;
        } catch (NumberFormatException e) {
        }

        try {
            fuelId = Integer.parseInt(fuelIdTFXML.getText());
            i++;
```

```
        } catch (NumberFormatException e) {
        }

        if (i == 0) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Заповніть хоч одне поле");
            alert.showAndWait();
            return;
        }

        int selectedIndex = table.getSelectionModel().getSelectedIndex();
        int eId = 0;
        try {
            eId = table.getItems().get(selectedIndex).getExpenseId();
        } catch (ArrayIndexOutOfBoundsException e) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Необхідно обрати поле для переписання");
            alert.showAndWait();
            return;
        }
        try {
            resultSet = null;
            resultSet = statement.executeQuery("select expenseId, fuelId, year, month,
amountOfSpend, reportId from expense where expenseId =" + eId);
            while (resultSet.next()) {
                if (expenseId == 0) {
                    expenseId = resultSet.getInt(1);
                }

                if (fuelId == 0) {
                    fuelId = resultSet.getInt(2);
                }

                if ((year * 100 + month) == 0) {
                    year = resultSet.getInt(3);
                    month = resultSet.getInt(4);
                }
                reportId = year * 100 + month;
                System.out.println(reportId);
                System.out.println(expenseId);

                if (amountOfSpend == 0.0) {
                    amountOfSpend = Double.parseDouble(resultSet.getString(5));
                }

            }
            statement.executeUpdate("update expense set expenseId = " + expenseId + ", fuelId =
" + fuelId + ", year = " + year + ", month = " + month + ", amountOfSpend = " + amountOfSpend +
", reportId = " + reportId + " where expenseId = " + eId);
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        initialize();
    }

    @FXML
    public void delete(ActionEvent event) {
        ObservableList <Expense> cells = FXCollections.observableArrayList();
        cells = table.getSelectionModel().getSelectedItems();
        if (cells.size() == 0) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Помилка!");
            alert.setHeaderText(null);
            alert.setContentText("Оберіть хоча б один запис");
            alert.showAndWait();
            return;
        }
        for (int i = 0; i < cells.size(); i++) {
            int expenseId = cells.get(i).getExpenseId();
            try {
                statement.executeUpdate("delete from expense where expenseId = " + expenseId);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        initialize();
    }
}
```

Додаток 3

Розробка програмного забезпечення для обліку використання енергоресурсів на об’єктах виробництва на платформі ОС Windows

Опис програмного модулю

УКР.НТУУ“КПІ”.ТР61117_20Б 13-1

Аркушів 6

2020

АНОТАЦІЯ

Метою дипломної роботи була розробка програмного забезпечення для ведення обліку використання енергоресурсів на об’єктах виробництва, яке би дозволило користувачу вести облік використання палив, вести звітну документацію у загальному та деталізованому вигляді.

В ході роботи було проаналізовано автоматизовану систему комерційного обліку енергоресурсів “АСКОЕ Пульсар”. В роботі було розроблено спрощений алгоритм комерційного обліку енергоресурсів. На основі розробленого алгоритму було створено програмну систему комерційного обліку енергоресурсів, яка є простою в використанні, обслуговуванні, використовує лише програмні модулі які розповсюджуються безкоштовно у вільному доступі та має простий і зрозумілий інтерфейс, що робить її більш універсальною та придатною для використання на потужностях малих виробництв, або навіть у домашніх господарствах.

В якості подальшого розвитку програмної системи планується створення модулю ручного вибору функціоналу, розширення спектру даних, що зберігаються та обороблюються та, загалом, розширення доступного інструментарію для обліку витрат енергоресурсів та підвищення рівня автоматизації системи.

ЗМІСТ

1. ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Даний програмний модуль розроблено у середовищі програмування IntelliJ IDEA, об’єктно-орієнтованою мовою програмування Java, із використанням системи управління базами даних MySQL та збірника проектів Apache Maven. Для створення інтерфейсу користувача було використано платформу JavaFX та візуальний модуль Scene Builder.

Програмна система призначена для ведення обліку використання енергоресурсів на об’єктах виробництва.

1.1 Опис логічної структури

Було розроблено кроссплатформену програмну систему ціллю якої є ведення місячного обліку використання палив на об’єкті виробництва.

Програмний продукт було створено у вигляді десктопного віконного додатку, який має віконний інтерфейс користувача.

Система складається із трьох основних частин: користувацького інтерфейсу, алгоритму обробки даних та бази даних розгорнутої на віртуальному сервері. Система керування базою даних потребує обов’язкового встановлення додаткового програмного забезпечення: серверу бази даних, середовища для керування базою та драйверу для підключення додатку до програмної системи.

Консольний додаток отримує дані як безпосередньо від користувача через візуальний інтерфейс, так і від бази даних через запити, опрацьовує отриману інформацію, використовуючи алгоритм обробки та передає користувачу таблицю наявних звітів із можливістю представлення деталізованої інформації по звіту.

1.2 Вхідні та вихідні дані

Вхідними даними для програмної системи є раніше введені дані про палива та витрати, що вже зберігаються в базі даних або аналогічні дані, введені безпосередньо користувачем в візуальному інтерфейсі системи.

Вихідними даними, що надає система є:

- Узагальнений звіт про використання палив в даному місяці;
- Параметризований та деталізований звіт про використання палив із деталізацією даних по кожному із видів палива;

2. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано на комп’ютері із процесором Intel(R) Core(TM) i3-5015U на базі архітектури x64 і тактовою частотою 2.10 GHz. Комп’ютер оснащений 4 Гб оперативної пам’яті із встановленою системою Microsoft Windows 10. Програмна система є кросплатформеним та не потребує великих витрат апаратних потужностей, тому може використовуватись на будь-якій іншій операційній системі із обов’язково попередньо встановленою Java SE та системою управління базами даних, а також може запускатися на більш слабких апаратних системах із незначним падінням швидкодії.